UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/816,465	03/26/2001	Alban Couturier	Q63703	6482

7590 10/07/2003

SUGHRUE, MION, ZINN, MACPEAK & SEAS, PLLC
2100 Pennsylvania Avenue, N.W., Suite 800
Washington, DC 20037-3213

EXAMINER	
HONG, HARRY S	

ART UNIT	PAPER NUMBER
2642	4

DATE MAILED: 10/07/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

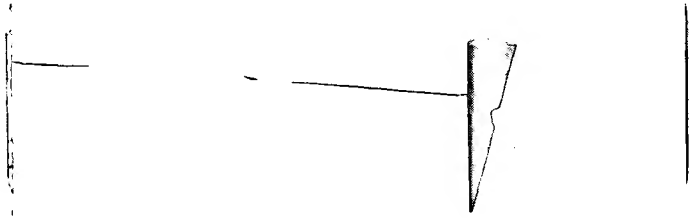
Organization TC2600 Bldg./Room PK2
U. S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
WASHINGTON, DC 20231
IF UNDELIVERABLE RETURN IN TEN DAYS

EQUAL OPPORTUNITY

OFFICIAL BUSINESS

WASHINGTON DC OCT 06'03
U.S. OFFICIAL MAIL
PENALTY FOR PRIVATE USE $300
PB METER #249720
03.95
U.S. POSTAGE

RETURN to SENDER



Office Action Summary	Application No.	Applicant(s)	
	09/816,465	COUTURIER, ALBAN	
	Examiner	Art Unit	
	Harry S. Hong	2642	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *26 March 2001* .

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-9* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☒ Claim(s) *1-8* is/are allowed.

6)☒ Claim(s) *9* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *26 March 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)☐ The proposed drawing correction filed on _____ is: a)☐ approved b)☐ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____ .

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) ☐ The translation of the foreign language provisional application has been received.

15)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) *3* .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

Priority

1. Receipt is acknowledged of papers submitted under 35 U.S.C. 119(a)-(d), which papers have been placed of record in the file.

Specification

2. The abstract of the disclosure is objected to because of the reasons below. Correction is required. See MPEP § 608.01(b).

3. Applicant is reminded of the proper language and format for an abstract of the disclosure.

The abstract should be in narrative form and generally limited to a single paragraph on a separate sheet within the range of 50 to 150 words. It is important that the abstract not exceed 150 words in length since the space provided for the abstract on the computer tape used by the printer is limited. The form and legal phraseology often used in patent claims, such as "means" and "said," should be avoided.

Claim Rejections - 35 USC § 112

4. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

5. Claim 9 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

6. Claim 9 recites the limitation "said master module" on line 4. There is insufficient antecedent basis for this limitation in the claim.

Claim Rejections - 35 USC § 102

7. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

8. Claim 9, as best understood, is rejected under 35 U.S.C. 102(b) as being clearly anticipated by Dickman et al. (Dickman; U.S. Patent No. 5,323,452; cited and applied for the first time) or Boyle, III et al. (Boyle, U.S. Patent No. 5,717,747; cited and applied for the first time).

Dickman plainly teaches adding a new call-processing module dynamically to a service call management system where the module reads on the nodes of Dickman and the identifier reads on the node name.

Boyle also plainly teaches adding a new call-processing module dynamically to a service call management system where the module reads on the new calling features of Boyle and the identifier reads on the feature name.

Allowable Subject Matter

9. Claims 1-9 are allowed over the prior art of record.

Conclusion

10. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. The patents to Mercouroff et al. teach call processing using CORBA specifications.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Harry S. Hong whose telephone number is (703) 306-3040. The examiner can normally be reached on Monday-Friday, alternate Fridays off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ahmad F. Matar can be reached on (703) 305-4731. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-4700.

Harry S. Hong

Harry S. Hong
Primary Examiner
Art Unit 2642

September 30, 2003

UNITED STATES DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450

APPLICATION NO./ CONTROL NO.	FILING DATE	FIRST NAMED INVENTOR / PATENT IN REEXAMINATION	ATTORNEY DOCKET NO.

EXAMINER

ART UNIT	PAPER
	4

DATE MAILED:

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner for Patents

Harry S. Hong
Primary Examiner
Art Unit: 2642

Form PTO-1449 (Rev. 2-32)	U.S. Department of Commerce Patent & Trademark Office	Atty. Docket No. Q63703	Serial No.: not yet assigned Confirmation No.: not yet assigned
INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)		Applicant: Alban COUTURIER	
		Filing Date: March 26, 2001	Group: not yet assigned 2642

U.S. PATENT DOCUMENTS

Examiner Initial		Document Number	Date	Name	Class	Sub-Class	Filing Date (if appropriate)

FOREIGN PATENT DOCUMENTS

		Document	Date	Country	Class	Sub-class	Translation Yes/No

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

HH		CORBAMED: "corbamed/99-04-044 Resource Access Decision", April 26, 1999, Object Management Group XP002153807

EXAMINER: HONG	DATE CONSIDERED: 9-30-03

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; draw line through citation if not in conformance and not considered. Include copy of this form with next communication.

Notice of References Cited	Application/Control No. 09/816,465	Applicant(s)/Patent Under Reexamination COUTURIER, ALBAN	
	Examiner Harry S. Hong	Art Unit 2642	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,323,452	06-1994	Dickman et al.	379/201.04
	B	US-5,717,747	02-1998	Boyle et al.	379/201.03
	C	US-6,201,862 B1	03-2001	Mercouroff et al.	379/230
	D	US-6,317,428 B1	11-2001	Mercouroff et al.	370/360
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

6

US006317428B1

(12) **United States Patent**
Mercouroff et al.

(10) **Patent No.: US 6,317,428 B1**
(45) **Date of Patent: Nov. 13, 2001**

(54) **METHOD OF PROVIDING A SERVICE TO USERS OF A TELECOMMUNICATION NETWORK, SERVICE CONTROL FACILITY, AND PROCESSING NODE**

(75) Inventors: **Nicolas Mercouroff**; **Alban Couturier**, both of Paris (FR)

(73) Assignee: **Alcatel**, Paris (FR)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/059,837**

(22) Filed: **Apr. 14, 1998**

(30) **Foreign Application Priority Data**

Apr. 14, 1997 (EP) 97 440 037

(51) **Int. Cl.**[7] **H04M 7/00**; H04L 12/66

(52) **U.S. Cl.** **370/360**; 370/352; 370/386; 379/207; 379/230

(58) **Field of Search** 370/259, 360, 370/352, 357, 384, 386, 387, 388, 522, 496; 379/201, 207–208, 219–221, 229–230; 706/10

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,754,546 * 5/1998 Viot et al. 370/384
6,205,212 * 3/2001 Swale 379/208

FOREIGN PATENT DOCUMENTS

9534175 12/1995 (WO) .

OTHER PUBLICATIONS

"Distributed Control Node Architecture in the Advanced Intelligent Network", M. Hirano et al., *XV International Switching Symposium*, Berlin, Apr. 23, 1995, pp. 278–282.

"CORBA: A Common Touch for Distributed Applications", F. Tibbits, *Data Communications*, vol. 24, No. 7, May 21, 1995, pp. 71–75.

"The Information Services Supermarket—an Information Network Prototype", I. Marshall et al, *BT Technology Journal*, vol. 13, No. 2, Apr. 1995, Ipswich GB, pp. 132–142.

"Distribution of Service Data to Distributed SCPs in the Advanced IN", N. Kusaura et al, *GLOBECOM 95*, Singapore, Nov. 14, 1995, pp. 1272–1276.

"Object–oriented Switching Software Technology", K. Maruyama, *IEICE Transactions on Communications*, Tokyo 1992, vol. E–75B, No. 10, pp. 957–968.

"Signalling in the Intelligent Network", M. Bale, *BT Technology Journal*, vol. 13, No. 2, Apr. 1995, Ipswich GB, pp. 30–42.

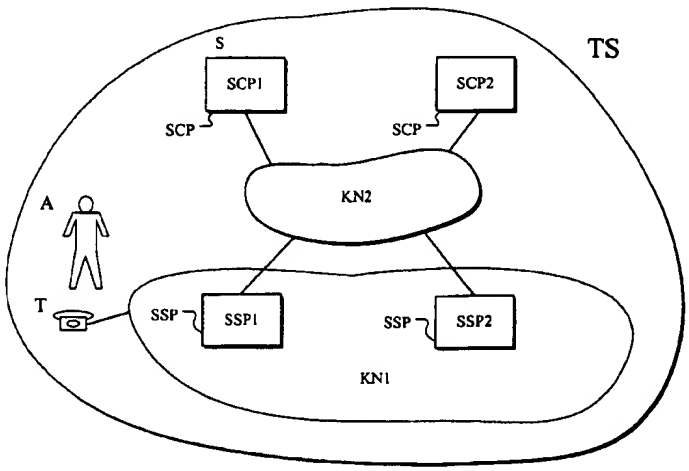
* cited by examiner

Primary Examiner—Wellington Chin
Assistant Examiner—Maikhanh Tran

(57) **ABSTRACT**

The invention concerns a method for providing a service for users of a telecommunication network. Service calls requesting the execution of the service for an respective one of the users are routed to a service switching exchange of the telecommunication network or are respectively routed to one of several service switching exchanges of the telecommunication network. A corespondent service request is sent by the respective service switching exchange, that has received the respective service call, to a service control facility or to one of several possible service control facilities. For each such service request received from the or each service switching exchange a service session object, which is able to interact and communicate via an object infrastructure with other objects in a object oriented computing environment, is created within the or each service control facility and the respective service session object controls the execution of the service for the respective service call.

18 Claims, 4 Drawing Sheets



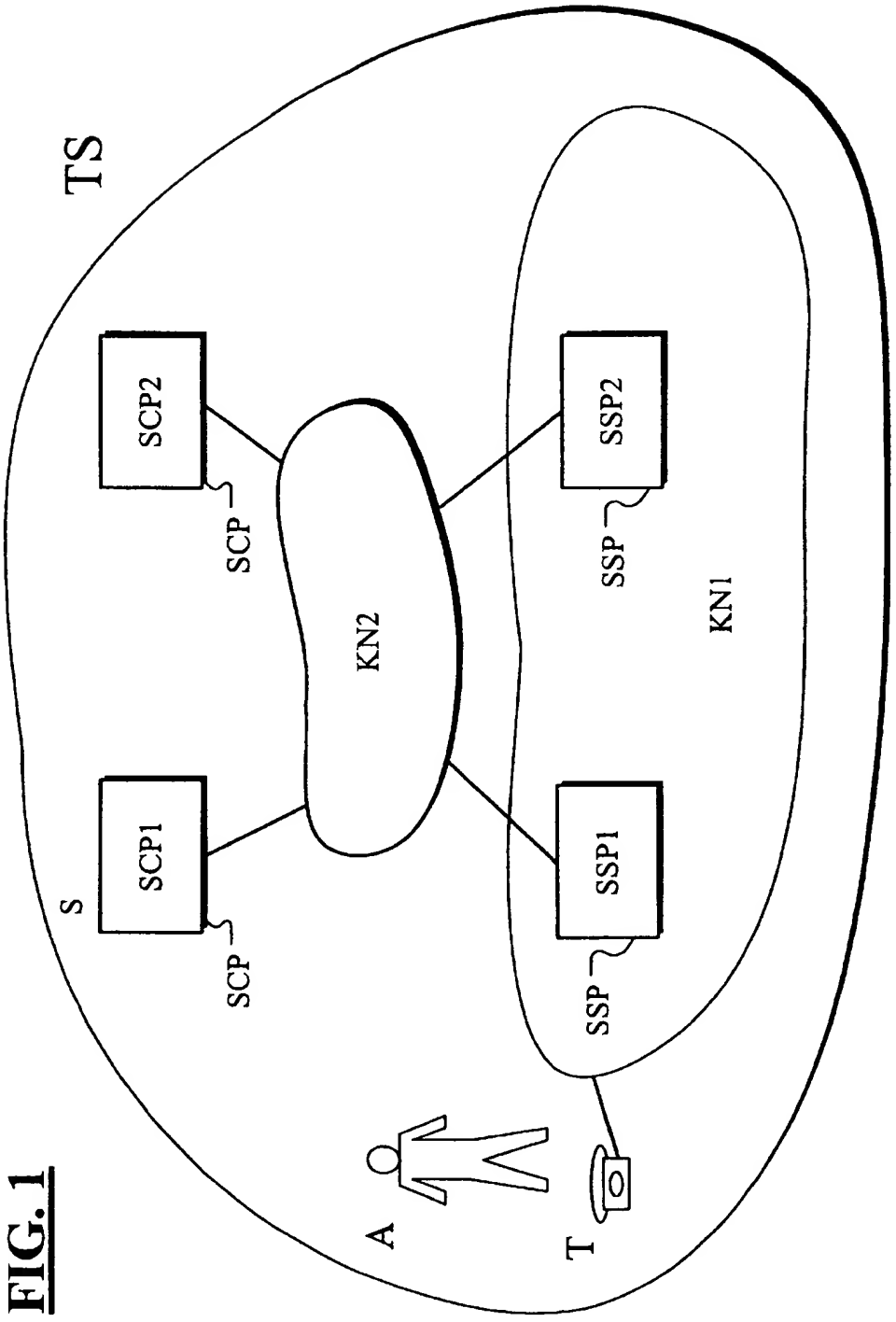


FIG. 1

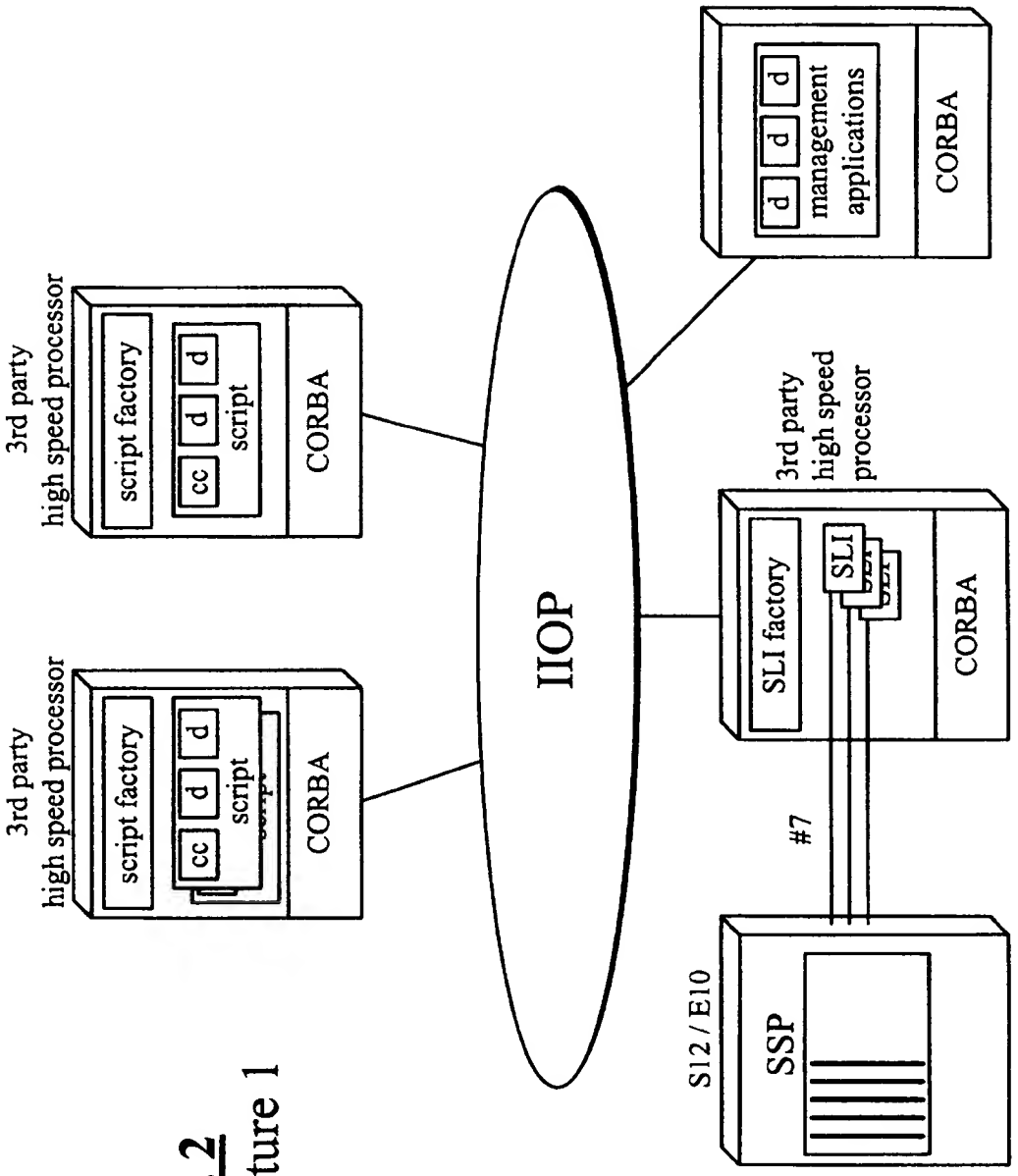


FIG. 2
Architecture 1

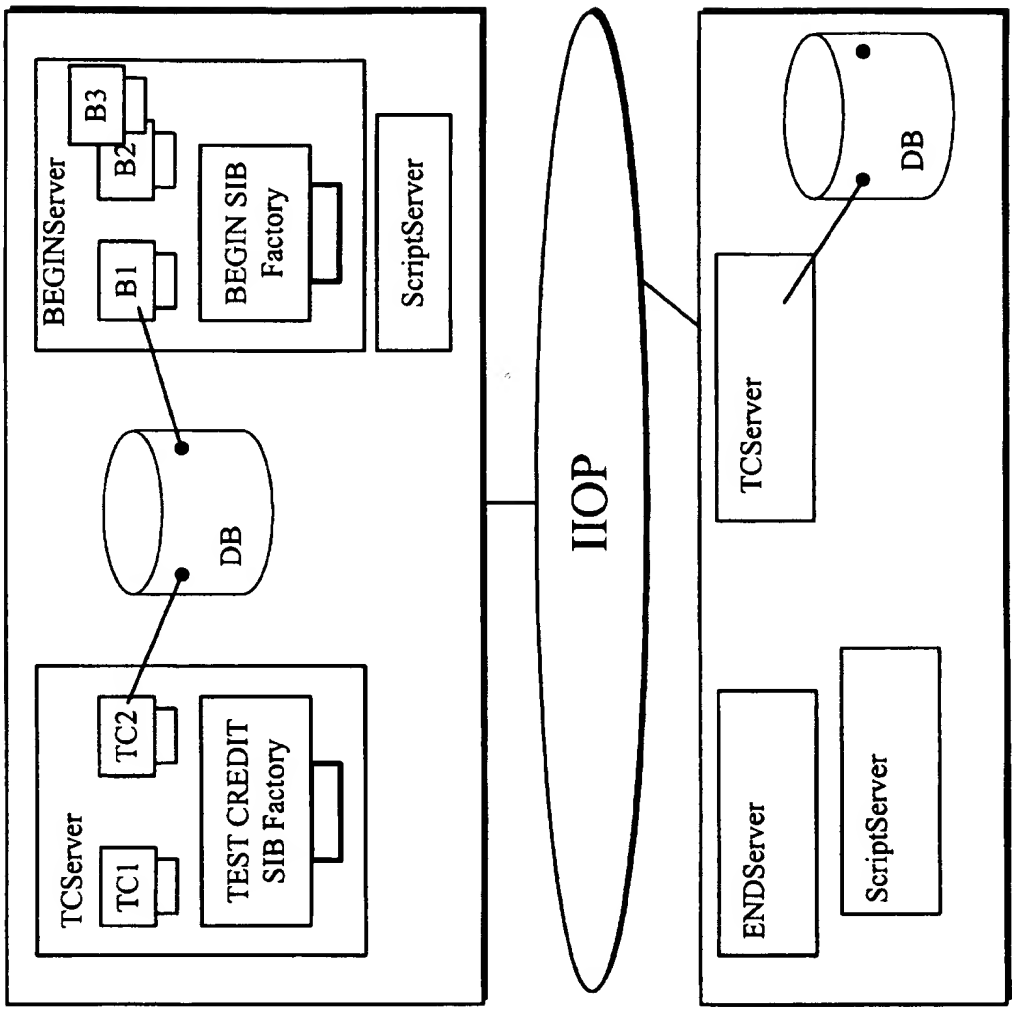


FIG. 3
Architecture 2

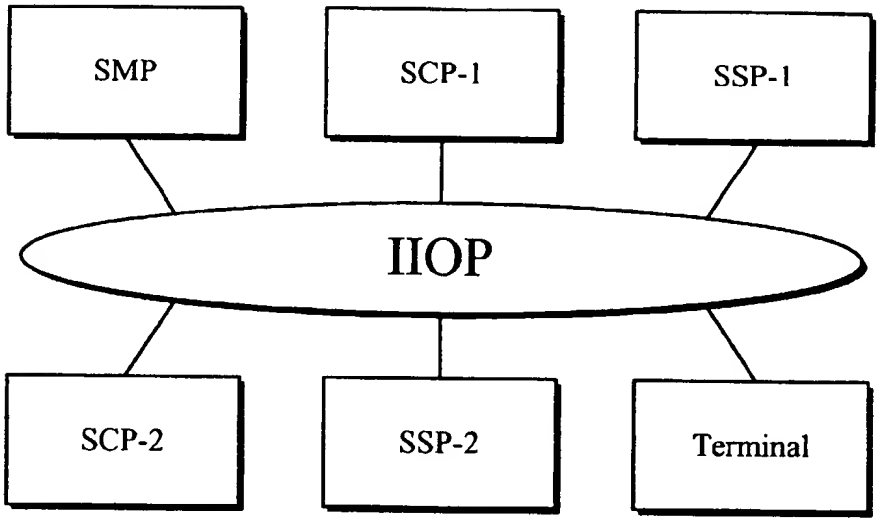


FIG. 4

Architecture 3

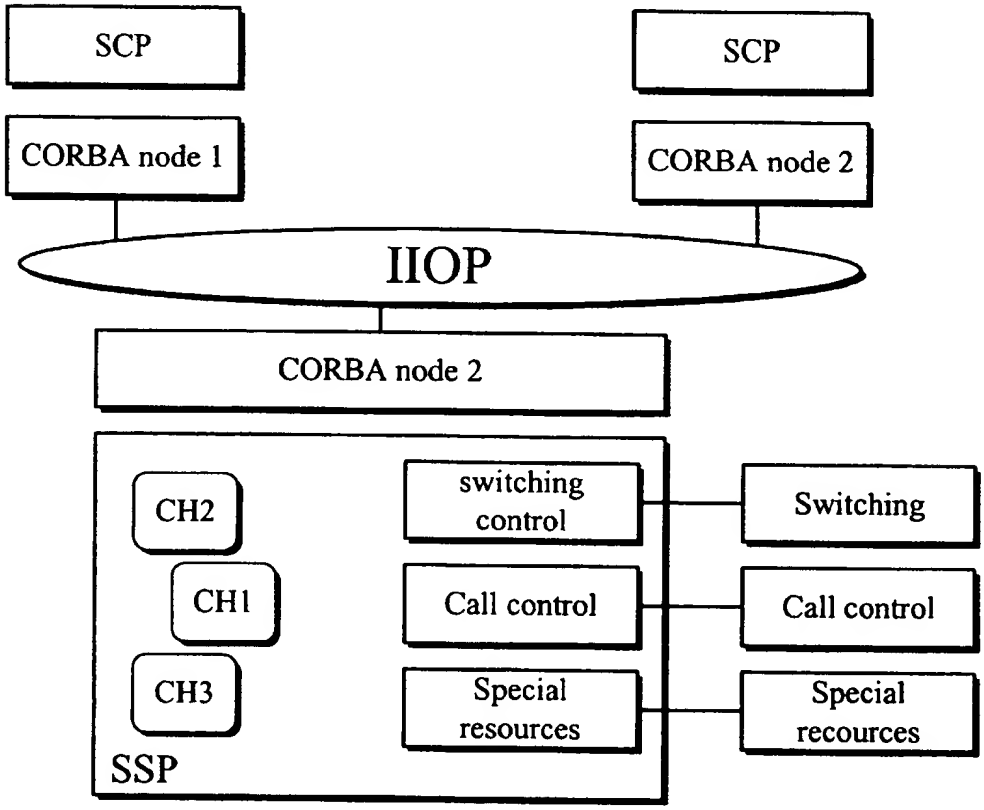


FIG. 5

SSP Server

METHOD OF PROVIDING A SERVICE TO USERS OF A TELECOMMUNICATION NETWORK, SERVICE CONTROL FACILITY, AND PROCESSING NODE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application discloses subject matter that is disclosed and which may be claimed in copending patent applications entitled "Method for Communicating Between a Service Switching Exchange of a Telecommunication Network and a Service Control Facility, filed Apr. 9, 1998 (U.S. Pat. No. 6,266,406 B1); and "Method of Providing at least One Service to Users of a Telecommunication Network, Service Control Facility and Server Node", filed on even date herewith (U.S. Pat. No. 6,201,862 B1); both of which are hereby incorporated by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention concerns a method for providing a service for users of a telecommunication network, a method for provisioning the execution of a service logic function within a service control facility of a telecommunication system, a service control facility for connecting to one or several service switching exchanges of a telecommunication network and a processing node for a service control facility connected to one or several service switching exchanges of a telecommunication network.

2. Discussion of Related Art

Telecommunication services can be provided according to the Intelligent Network Architecture.

A user of a telecommunication network requests a service by dialing the number dedicated to the service. The call with this number is routed through the telecommunication network to a service switching exchange which recognizes this call as a call requesting the service. Then, this service switching exchange communicates via a data network with a service control facility, the so called service control point. This service control point contains a service logic which contains a description of the method that has to be executed to provide the service to the calling user. By a request of the service switching exchange the execution of this method by the service logic is initiated and the requested service is provided to the requesting user.

This service logic is realized as a single process handling all calls one after the other. Each of them going through a single queue. Each call is associated with a context allowing the state of the call not to be lost between user interaction. The service is executed through a finite state machine, taking as input the TCAP message and the context of the call.

The disadvantage of this approach is that this architecture of provisioning services is that within the service control facility an incoming request for a service has to wait for execution until the current one has been processed out. Therefore, the rate of service requests that can be handled by the service control facility is strictly limited.

SUMMARY OF INVENTION

Accordingly, it is a primary objective of the present invention to increase the number of service requests that can be handled by a processing node that is involved in the execution of services for users of a telecommunication network.

According to a first aspect of the invention, a method for providing a service for users of a telecommunication network, in which method service calls requesting the execution of the service for an respective one of the users are routed to a service switching exchange of the telecommunication network or are respectively routed to one of several service switching exchanges of the telecommunication network and in which method a corresponding service request is sent by the respective service switching exchange, that has received the respective service call, to a service control facility or to one of several possible service control facilities, is characterized in that for each such service request received from the or each service switching exchange a service session object, which is able to interact and communicate via an object infrastructure with other objects in a object oriented computing environment, is created within the or each service control facility and the respective service session object controls the execution of the service for the respective service call.

According to a second aspect of the invention, a method for provisioning the execution of a service logic function within a service control facility of a telecommunication systems, where service requests, that request the execution of the service logic function, are received by the service control facility from at least one service switching exchange of the telecommunication system, is characterized in that for each such service request received from the at least one service switching exchange a service session object, which is able to interact and communicate via an object infrastructure with other objects in a object oriented computing environment, is created within the service control facility and the respective service session object handles the execution of the service logic function for the respective service request.

According to a third aspect of the invention, a service control facility for connecting to one or several service switching exchanges of a telecommunication network, where the service control facility contains means for receiving service requests, that request the execution of a service for a user of the telecommunication network, from at least one of the service switching exchanges of the telecommunication network, is characterized by containing means for creating for each such service request received from the at least one service switching exchange a service session object within the service control facility, means for enabling each service session object to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and means for executing under control of the respective service session object a service logic function for the respective service request.

According to a fourth aspect of the invention, a processing node for a service control facility connected to one or several service switching exchanges of a telecommunication network, where the processing node contains means for receiving service requests, that request the execution of a service for a user of the telecommunication network, from at least one of the service switching exchanges of the telecommunication network, is characterized by containing means for creating for each such service request received from the at least one service switching exchange a service session object, means for enabling each service session object to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and means for executing under control of the respective service session object a service logic function for the respective service request.

The underlying idea of this invention is that for each service request received from a service switching exchange a service session object, which is able to interact and

communicate via an object infrastructure with other objects in an object oriented computing environment, is created within the control facility and the respective service session object controls the execution of the service for the respective service call. Therefore, the service architecture is no longer based on a functional design and technologies like multi-threading or multi processing can be applied.

Due to the introduction of an object oriented computing environment and the special object design described above, the processing of the service requests can be distributed and this proved high scalability and IT platform independence. The redesign of the service logic according to this approach allows for easy service interworking personalization, distribution and maintainability of the software. The design pattern allows for taking all benefits from object oriented programming.

Furthermore, it allows introduction of multithreading. The direct benefit is that a service session, that means the processing of a service request, is not blocked by the execution of another one.

The use of a factory allows implementation of several life cycle policies for the service session objects.

It is further advantageous

- to implement each service by a dedicated CORBA server. Each call to the service is handled then by a single CORBA object;
- to manage the creation of a service session object by a service dedicated factory;
- to set the interface definition of a service session object as TCAP mapped over IDL.

By the use of CORBA (Common Object Request Broker Architecture) servers the service control facility implementation is simplified and scalability is ensured.

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** is a block diagram presenting the architecture of a telecommunication system containing a service control facility according to the invention.

FIG. **2** is a block diagram presenting a first object architecture of the telecommunication system.

FIG. **3** is a block diagram presenting a second object architecture of the telecommunication system.

FIG. **4** is a block diagram presenting a third object architecture of the telecommunication system.

FIG. **5** is a block diagram presenting a object architecture of a service switching exchange for the telecommunication system.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. **1** shows a telecommunication system TS with an telecommunication network KN1, a subscriber terminal T assigned to a subscriber A, a communication network KN2 and two service control facilities SCP1 and SCP2.

The telecommunication network KN1 is for example a telephone network. It is also possible that this network is a data network or a communication network for mixed data speech and video transmission.

From the exchanges of the network KN1 two special designed exchanges SSP1 and SSP2 are shown. These exchanges are service switching exchanges that contain a service switching functionality SSP. To these exchanges service request calls from the subscriber of the network are routed. When such a service switching exchange receives such a service request, it sends a corresponding request via the network KN2 to one of the two service control facilities SCP1, SCP2. The service is then provided under the control of the respective one of the service control facilities SCP1 and SCP2.

Each of the service control facilities SCP1 and SCP2 provides one or several services S and realize a service control function.

The communication network KN2 is preferably a data network, especially according to the SS7 signaling protocol. It is also possible that this network is a LAN (Local Area Network), a MAN (Metropolitan Area Network) or an ATM (Asynchronous Transfer Mode) network.

Preferably, the service switching exchanges SSP1, SSP2, and the communication between them and the service control facilities SCP1, SCP2 are designed according to the Intelligent Network Architecture (according to the service switching point and according to the communication between the service switching point and the service control point), described for example in the ITU-T Q.1214 Recommendation.

If a user wants to start an service S, he dials a specific number for that service. The local exchange will route this call to the nearest service switching exchanges and this will trigger the service switching functionality SSP to start the service. This is done by sending a request to the SCP to start executing a Service Logic Program (SLP). An SLP is specific for each service, it will instruct the network on how to handle the service. It will execute service logic, statistic logic, database logic, charging logic, etc. It will also provide instructions for the switch (e.g., create the second leg of the call) and Intelligent Peripherals (IP) (e.g., announcement machines). A protocol according to the Information Networking Architecture (INAP) is used for this, and INAP messages are transported in TCAP (Transactional Capabilities Application Part) messages between the SCP and SSP.

When designing a distributed application, first the possible system components to be distributed are identified (i.e., what the objects of the system are and how they will be grouped into larger objects that will be distributed).

The breaking down of the SCP into objects offers some possibilities. FIGS. **2** to **3** depict these organized into different architectures. All state information for handling one call was kept together and that all needed data was organized into objects. So, one possibility for SCP objects is a script object that executes the service logic for one call and an object per data object.

The values of the data objects are kept in a database, so having data objects that hide the use of a database is preferable (i.e., certain independence on the database). The catch lies in the fact that making these objects CORBA objects gives us a considerable overhead when methods (e.g., to read or write attribute values) are called on them. So, the system performance would become unacceptable.

The script object could be decomposed into a service execution engine and an SIB graph. So, SIBs could become distributed objects too. If database objects are distributed, this offers the possibility to execute an SIB on the machine where the particular data object resides. We can envisage a scheme where there is a control entity (a script) that executes SIBs remotely (possibility of having more overhead) or a scheme where control is passed from SIB to SIB. This trade off between getting the data and running a script on one machine and running parts of a script (SIBs) where the data resides is certainly worth investigation.

The SSPs main functionality is to provide a hook between the call processing and the IN system. An SSP server must

at least have an object that bridges between the normal Call Control Function (CCF) and an object that bridges between the Switching Function (SF). Also an object that bridges between Specialized Resources (SRF) (e.g., announcement equipment), seems to be obvious. In FIG. **5**, these objects are depicted as a Switching Control, Call Control and Special Resources Object. We also need an object, Call Handler (CH) that interlaces with the SCP and the other objects in the SSP. There are two possibilities: there will be only one that handles all calls, or several, each handling one call. If we have several, existing only for the duration of the call, we also need a corresponding factory object to control the lifecycle of these objects.

In the following is given the architecture of an SSP server on CORBA.

For the Switching Control, Call Control and Special Resources objects we could have two approaches. In the first approach they are only interface objects between the proprietary Application Program Interface (API) of existing software. In the second approach they would control the hardware directly, what implies providing a CORBA interface to Switching Control, Call Control & Special Resources. In theory this would be the best solution, since the approach would give an overall consistent software architecture, in this case special IDL (Interface Definition Language) adapter interfaces have to be designed.

An SMP (Simple Management Protocol) is responsible for service control and monitoring. In order to be able to control services, the SMP does not need to be a DPE object. What is needed is that the objects that are controlled (e.g., script) provide an interface to do so. For monitoring we need an SMP object that is able to receive "events" from the objects that are being monitored. In our research prototype we defined and implemented a trace control as an example on what is possible as management functionality on the SCP.

In the previous section we have described a number of possibilities for CORBA objects in an IN Architecture. In this section we will group certain of these CORBA objects into a specific architecture. The sequence of the architectures can be seen as a possible evolution scenario, starting from current day's service provisioning, especially IN, products.

The basis of the first architecture is the usage of a distributed database for data distribution and CORBA distribution for distributed SCP. This architecture is designed to interface with existing SSPs, so the SSP is not a CORBA object. Since this architecture was used to implement a research prototype, we elaborate on it more than the others. FIG. **2** depicts the architecture.

The basic components in the architecture are:

the script: a CORBA object that handles exactly one call

the SLI (Service Logic Interface), a CORBA object that handles a TCAP dialogue for, and interfaces with the SSP.

the distributed DB.

Existing IN service scripts can encapsulated in Script objects.

A Script object also contains the Service Data and Call Context related to a particular call, and one Script object is instantiated per call. The interface of the Script object contains operations that represent the TCAP messages that the Script receives during its execution.

A special Service Logic Interface (SLI) object receives the #7 messages from the SSP and dispatches them to the corresponding script object instantiation, using the ORB services. The interface of the SLI object contains the TCAP messages that can be sent to it. One SLI object is instantiated per call.

Since the Script and SLI objects exist only during the lifetime of a call corresponding factory objects are used to create and destroy them. Data objects are not CORBA objects, they are implemented as C++ objects. This approach hides the use of a particular database from the service logic.

It is possible to use as database for example Oracle, SQLNET or ObjectStore. The SMP has not be considered; we could have used the proprietary mechanism to send status information to it, or defined it as a CORBA object with an appropriate IDL interface.

However as an example on what is possible, its easy possible to design and implement a small trace control system; each server has a tracer object, the trace control part of the SMP can call methods to turn traces on particular implementation objects on and off. The SMP also manages the database (using SQLNET).

A script and a SLI are CORBA objects, how they are distributed over servers is a deployment issue. A CORBA server is a process running at a particular node in the system. There is one SLIServer in the system that interfaces with the SSP. It has one CORBA object, the SLIFactory, that is always present when the server is running.

The main purpose of this object is to create new SLI objects when new calls are started (when a TCAP dialogue is opened). The SLIServer is deployed in the node that has the #7 hardware and software installed. The system can have a number of ScriptServers, one per available node.

This server also has a factory object, that exists during the lifetime of the server itself. This factory object is registered in the CORBA Naming service, so that an SLI can find it. In this way service logic distribution is facilitated. Adding more processing power to the system just means installing the new machine, running the ScriptServer on it, and the next time the SLI looks for a ScriptFactory a new possibility is available. The same mechanism also provides some reliability, when a machine goes down for some reason, the SLI will simply not get a reference to that ScriptServer any more. Note that the ScriptFactory is also to best candidate to put a management interface on (e.g., to change the service characteristics), since it is always available as long as the server is running.

The interface between SLI and Script is defined to be asynchronous and in terms of TCAP primitives (BEGIN, INVOKE, RESULT, . . .). This option is advantageous, since one of the requirements is to reuse existing IN product software. Another option is to define this interface in terms of INAP primitives.

The difference between a CORBA object, a (CORBA) server and a process must be clear. A CORBA object implements an interface and runs within a CORBA server. The CORBA server itself is one process that runs on a host (or node). This definition is important for the relation between a script and a call. In the current ALCATEL IN product implementation a script is a process that handles different calls, and keeps a different call context for each of those. This is done for performance reasons, starting up a separate process (script) for each call takes some time.

We could adapt a similar approach (one script handles n calls) in these architectures, but the fact that a script has multiple contexts is not that clean and it is possible to deal with the performance issue by having a CORBA server with multiple scripts (each handling one call). So, it is better that in this architecture, a host can run one or more CORBA servers, each running different script objects that each handle a specific call.

The CORBA server is similar to the script in the current IN product, but the problem of handling multiple contexts is shifted from the application programmer to the CORBA platform.

A step further in this sequence of architectures, as second architecture, is to break down the script object into its components. As known, IN services are designed as a number SIBs that are nodes in a service logic graph. The execution of the script means executing SIBs and moving to the next SIB depending on data and responses coming from the SSP. In stead of designing this as a monolithic piece of software (from the SCE), it could be designed in terms cooperating SIBS. As known, the IN capability set 1. SIBs are not sufficiently defined to used them as they are. This resulted in IN platform providers having their own set of SIBs derived from the standard and software from SIBs is not reusable among platforms.

When SIBs are defined as CORBA objects this situation could be broken and service providers would be able to design their own services, reusing existing SIBs (source code).

Having SIBs as independent CORBA objects offers the possibility to execute the SIBs in different places. Instead of getting the data (using a distributed database, or using CORBA objects) to the place where the script or SIB is executed, we could start the SIB on the machine where the data is located. This brings benefits. We can envisage a scheme where there is a control entity (a script) that executes SIBs remotely or a scheme where control is passed from SIB to SIB.

A third architecture breaks with traditional IN product in the sense that it also makes the SSP a CORBA object. In the previous architectures, for reliability reasons, it is still necessary that CORBA servers are deployed on machines that are interconnected on a reliable LAN. In practice, with the current DPE implementations (TCP/IP) this means putting the machines in the same location. This technology restriction prevents putting the SSP on the DPE, since in practice SCP and SSP sites are located on different places. However, technology evolves and DPE providers (e.g., IONA) are looking into the possibility of porting their product to other transport protocols, e.g., #7 links for telecommunication purposes. When this happens the step to making the SSP a CORBA object is not that big. In this case, the SLI object would not be needed anymore, since the SSP object could take over its role. When a call is started it gets a reference to a ScriptFactory and asks to create the script object itself. It could then communicate with this object directly. Also the language used in this communication (INAP encapsulated in a TCAP based interface) can be simplified. This interface could be defined in terms of INAP primitives (e.g., provide_instruction(args), join(args)). This would improve (simplify) the system design and implementation, since the TCAP layer disappears for the application programmer. One could say that when CORBA is used, the application programmer must only be concerned with the application layer of the communication, not with the session layer (when TCAP is used).

The final step would be to extend integrate a terminal (as proposed by TINA (Telecommunications Information Networking Architecture)). This would mean a major change in user equipment. However this change is not infeasible if we consider the fact that more and more users get connected to the internet over their telephone lines. And the technology needed to have the DPE extended to the terminal is similar to this. The benefit of this would be that the terminal to network protocol could become much richer than just passing DTMF tones as numbers.

The above described architecture provides a solution to make especially IN platforms more scalable. Since interworking with and evolution from existing products is important we presented a sequence of architectures that can be seen a possible evolution scenario.

As example an implementation procedure according to the architecture 3 is described in the following:

A Credit Card Calling service which existed for the UNIX based IN Release existing Alcatel product is to be implemented. The Credit Card Call service is a "typical" IN service and allows a call from any terminal to be billed to a particular card number. To set up that call, the user has to enter a card number, a PIN code and the destination number. If the entered data is correct, the call is completed to the destination address and charged to the user's card.

An SSP simulation stub was used to generate the TCAP dialogue for the Credit Card Call. The SSP stub is a CORBA object that sends and receives the same TCAP messages that an SSP would. The SSP stub is capable of simulating different call scenarios, can generate different call loads and can repeat a scenario an arbitrary number of times.

Another general issue for the use of CORBA in an IN oriented service provisioning architecture is the interworking of CORBA-based IN with legacy IN applications.

This interworking can be achieved with the use of Adaptation. It is related to the way a CORBA based infrastructure can communicate with the legacy of switching systems, namely SSPs.

There are two approaches possible:

1) Definition of an Adaptation Unit which is an application level bridge that provide the mapping of SS7 and TCAP primitives into IDL invocations. This approach is similar to one taken by the XoJIDM group for the CORBA/CMIP gateway. The result of their works can be reused especially the static mapping of GDMO/ASN.1 to IDL interfaces.

In order to minimize the impact on existing systems, CORBA should provide framework services and tools in order to achieve this mapping. The availability of such framework will allow interworking of CORBA-based IN with a variety of existing hardware such as SCPs and HLRs.

Such application level bridge should be characterized by high availability and high performance without being a bottleneck for a distributed system. However for the required real-time performance, this is an intermediate solution towards full IN CORBA-based systems.

This approach would involve building an IDL interface to represent application level protocols such as MAP and INAP and others which are based on the TCAP layer. The TCAP layer provides a "ROSE-like" asynchronous RPC service over the SCCP layer. And basing the bridge on TCAP will exclude the use of circuit related signaling protocols such as ISUP and TUP from the solution.

Like in the XoJIDM approach, there should be a static translation of the INAP/TCAP specification to IDL interfaces which will be done by a dedicated compiler. Thus, any INAP dialect can be converted to appropriate IDL interfaces. These applications level bridges would implement these IDL generated interfaces and dynamically perform conversion between IDL-derived types and their ASN.1 equivalents.

CORBA nodes using the protocol specific bridges would generally run two servers, one each for processing outgoing and incoming invocations. Since TCAP is a connectionless service the gateway will have to maintain state in order to match invocations to responses and exceptions.

2) Usage of SS7 as an Environment Specific Inter-ORB protocol (ESIOP):

A possible solution is to map the CORBA GIOP on top of SS7 or to build a new Extended protocol (ESIOP) on top of the SS7 layers.

The ESIOP solution would essentially use an SS7 protocol as a transport for the GIOP protocol. CORBA objects

would be visible across the SS7 network in exactly the same manner as they would be across the Internet with the CORBA IIOP. This would allow as ORB to use existing SS7 infrastructure as transport.

It should be noticed that existing signaling networks were never dimensioned to support the sort of traffic which will be exchanged by CORBA nodes. However, there is a potential benefit in this approach by exploiting the fault tolerant nature of the SS7 network.

The first issue here is the choice of SS7 protocol access level for this solution. The two principal choices are TCAP and SCCP: GIOP at TCAP level:

It should be possible to use TCAP for carrying GIOP messages since IT is essentially a ROSE-like service. Services which make use of TCAP must define their operations using ASN.1 modules. It is envisaged that ASN.1 macros would be used to define the operation set: Request, Reply, CancelRequest, LocateRequest, LocateReply, CloseConnection and MessageError. These operations correspond to the GIOP primitives. The gateway would be responsible for converting CDR format, of the form used by GIOP, into a form transportable using ASN.1 types (possibly as an opaque buffer with BER encoding).

While it should be possible in principle to use TCAP as the basis for the ESIOP, it is not suitable because of:

the complexity of the implementation.

the overhead incurred in by the TCAP layer in addition to basic transport

the asynchronous nature of the protocol.

ESIOP at SCCP Level:

The other choice for implementing the SS7 ESIOP is at the SCCP level. The SCCP provides services corresponding to the OSI-RM network layer functionality. It can operate in both connection-oriented and connectionless mode. It should be feasible to use the SCCP to transport GIOP messages although the ESIOP code may be required to perform its own transport layer functions (e.g., end-to-end flow-control and segmentation/re-assembly). It should be possible to address CORBA nodes using the "Global Title" mode of SCCP addressing. It would appear that using SCCP as the basis for an ESIOP for the SS7 network would be the best approach.

The following requirements are advantageous for a object oriented environment for an infrastructure in which a service session object interacts (described by hand of the CORBA environment):

in terms of functional requirements, CORBA should provide:

asynchronous Invocation model and support for group communication in the ORB;

proper hooks for extending the ORB with security, transaction and replication mechanisms;

node management and minimal object lifecycle facilities;

a time service and native monitoring facilities for capturing both computational and engineering events;

support for multithreading with flexible event-to-thread mappings.

The non-functional requirements of CORBA are concerned with:

identifying the level of scalability and object granularity that the ORB should meet in the IN context;

identifying the performance level that the ORB must achieve in terms of latency, throughput, etc.;

providing a predictable ORB core by instrumenting and documenting the time behavior of the platform;

reliability which denotes requirements that define acceptable behaviors for the distributed applications in the presence of hardware and software failures. In this case two types of reliability can be identified; the integrity state and the availability;

manageability which denotes requirements that enable ease of development, deployment and operation for complex applications (in this case maintainability, (re) configurability and extensibility of CORBA applications.

An asynchronous Invocation model commonly used by IN applications is required. Thus, asynchronous and optionally isochronous invocation model is required with the ORB. The requirement here is a client side programming model which potentially supports a lightweight asynchronous communication mechanism incorporating a mandatory callback (or ORB "upcall" type mechanism) and optionally a "Futures" type programming model.

For the same object, both asynchronous and synchronous models should co-exist. The maximum concurrency level is defined in the configuration (e.g., QoS parameters), and has to be controllable, with the possibility of queuing the request or returning an exception.

A basic fault detection support should be provided by the ORB runtime for any object which needs it. This can be done by a timer which is implicit set and managed in the client process.

Flexibility and Scalability:

The ORB should be able to support applications handling a large number of objects and should be able to support many simultaneous connections to remote objects.

The memory cost of an unused remote interface reference in a given capsule (i.e., Unix process) should be of the order of a standard language pointer.

A typical telecommunications environment comprises objects of very different granularities in both space (memory size) and time (object lifetime and duration). A scalable ORB should support objects at different levels of granularity, and minimize the overhead associated with the handling of small and volatile objects and of connections to remote objects.

To achieve some of the ORB fault-tolerance and reliability, the CORBA object model could be enhanced to support groups of objects or globally known as group communication such as those described.

Reliability and Availability:

To achieve the reliability requirements, the notion of replicated distributed objects can be introduced in CORBA. Thus the critical components of the application are implemented through replicated objects. This replica management can be handled automatically by the ORB such that clients interacting with a given server object is not aware if it is replicated or not. The degree of replication can be determined by the desired level of reliability which can be measured with the number of maximum number of concurrent failures; the nature of the failure which can be typed (malicious or benign); and the type of reliability property being sought such as integrity or availability. High availability or Fault Tolerance capabilities should be provided by CORBA for distributed IN systems in order to ensure the same robustness as current IN systems.

Timeliness requirements can be also achieved by the replica service. For example, applications that need to have bounded and predictable delays will be implemented through replicated objects. Each replica is able to process locally all of the methods defined for the object. In the absence of failure, a client can use any of the responses to

a method Invocation as its result, (the invocations in this case is perform synchronously).

To achieve this, the ORB will have a modular structure allowing the implementation of different profiles, depending on application or service requirements. These profiles give an abstraction of the resources provided by the underlying operating system. One of the ORB modules will be a failure detector which is at the lower level of the ORB structure. And a response Invocation delay beyond a certain threshold is classified as failure. Thus, the client is able to trade off reliability for timeliness, the shorter the threshold, the tighter the bounds on delays. By replicating an object in a sufficient number of times, the client is able to meet both timeliness and reliability requirements simultaneously.

Here, we have identified a requirement for a replication service with correctness, safety and liveness properties; and a failure detector module which is part of the ORB core.

Performance:

Performance of IN are measured in number of calls per second for service nodes and intelligent peripherals; and number of transaction per second for signaling control point. These calls and transactions may involve multiple messages exchanged between an SSP and the Intelligent Layer.

To obtain the actual performance of legacy IN systems, real-time performance of the ORB is required for the use of distributed processing in IN systems as well as its distribution on a geographical scale (non centralized IN systems). To achieve better performance for IN distributed systems, the ORB call overhead should be reduced, and the performance level that the ORB must achieve in terms of latency, throughput, etc., should be defined and documented.

What is claimed is:

1. A method of providing a service for users of a telecommunication network, wherein service calls requesting execution of the service for a respective one of the users are routed to one respective service switching exchange of the telecommunication network, and wherein a corresponding service request is sent, by the respective service switching exchange that has received one of the service calls, to a respective service control facility, comprising the following steps:

creating within the service control facility a service session object for each of the service requests, and

controlling the execution of the service for the respective service call, wherein the service session object is able to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and wherein the execution of the service is controlled by the respective service session object.

2. A method as claimed in claim 1, characterized in that the service switching exchange communicates with the service control facility according to the communication protocols of the Intelligent Network Architecture.

3. A method as claimed in claim 1, characterized in that the service session object interacts and communicates as a CORBA object via a CORBA object infrastructure with other objects for controlling the execution of the service.

4. A method as claimed in claim 1, characterized in that the service control facility controls and carries out the execution of different services for users of the telecommunication network.

5. A method as claimed in claim 1, characterized in that the creation of each service session object is managed through a factory object.

6. A method as claimed in claim 1, characterized in that the creation of each service session object is managed by a service dedicated factory object.

7. A method as claimed in claim 5, characterized in that the factory object implements a life cycle policy for the service session object.

8. A method as claimed in claim 7, characterized in that the factory object implements a creation on demand life cycle policy for the service session object.

9. A method as claimed in claim 7, characterized in that the factory object implements an activation on demand life cycle policy for the service session object.

10. A method as claimed in claim 1, characterized in that all functions for controlling the execution of the service are implemented by a dedicated CORBA service server.

11. A method as claimed in claim 1, characterized in that the service session object has an interface definition which is Transactional Capabilities Application Part mapped over Interface Definition Language.

12. A method as claimed in claim 1, characterized in that the service session object directly receives its own message invocations.

13. A method as claimed in claim 1, characterized in that the service session object receives Information Networking Architecture messages as message invocations from the service switching exchange.

14. A method as claimed in claim 1, characterized in that the service session object runs in its own thread.

15. A method for provisioning execution of a service logic function within a service control facility of a telecommunication system, wherein service requests that request execution of the service logic function are received by the service control facility from at least one service switching exchange of the telecommunication system, comprising the steps of:

creating within the service control facility a respective service session object for each of the service requests received from the at least one service switching exchange, and

handling the execution of the service logic function for each of the service requests, wherein the service session object is able to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and wherein the execution of the service logic function is handled by the respective service session object.

16. A service control facility for connecting to one or several service switching exchanges of a telecommunication network, where the service control facility contains means for receiving service requests, that request the execution of a service for a user of the telecommunication network, from at least one of the service switching exchanges of the telecommunication network, characterized by containing means for creating for each such service request received from the at least one service switching exchange a service session object within the service control facility, means for enabling each service session object to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and means for executing under control of the respective service session object a service logic function for the respective service request.

17. The service control facility according to claim 16, characterized by containing a variable number of processing nodes for processing service session objects.

18. A processing node for a service control facility connected to one or several service switching exchanges of a telecommunication network, where the processing node contains means for receiving service requests, that request the execution of a service for a user of the telecommunica-

tion network, from at least one of the service switching exchanges of the telecommunication network, characterized by containing means for creating for each such service request received from the at least one service switching exchange a service session object, means for enabling each service session object to interact and communicate via an object infrastructure with other objects in an object oriented computing environment, and means for executing under control of the respective service session object a service logic function for the respective service request.

* * * * *

US006201862B1

(12) **United States Patent**
Mercouroff et al.

(10) **Patent No.: US 6,201,862 B1**
(45) **Date of Patent: Mar. 13, 2001**

(54) **METHOD FOR PROVIDING AT LEAST ONE SERVICE TO USERS OF A TELECOMMUNICATION NETWORK, SERVICE CONTROL FACILITY AND SERVER NODE**

(75) Inventors: **Nicolas Mercouroff**; **Alban Couturier**, both of Paris (FR)

(73) Assignee: **Alcatel**, Paris (FR)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/059,851**

(22) Filed: **Apr. 14, 1998**

(30) **Foreign Application Priority Data**

Apr. 14, 1997 (EP) 97440036

(51) **Int. Cl.**[7] **H09M 7/00**
(52) **U.S. Cl.** **379/230**; 379/221; 379/207
(58) **Field of Search** 379/201, 207, 379/219, 220, 221, 229, 230

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,455,854	*	10/1995	Dilts et al.	379/93.17 X
6,038,301	*	3/2000	Nightingale	379/201
6,052,456	*	4/2000	Huang	379/201

FOREIGN PATENT DOCUMENTS

9534175 12/1995 (WO) .

OTHER PUBLICATIONS

"Distribution of Service Data to Distributed SCPs in the Advanced IN", N. Kusaura et al, *Globecom 95*, Singapore, Nov. 14, 1995, pp. 1272–1276.

"Object–oriented Switching Software Technology", K. Maruyama, *IEICE Transactions on Communications*, Tokyo 1992, vol. E–75B, No. 10, pp. 957–968.

"Signalling in the Intelligent Network", M. Bale, *BT Technology Journal*, vol. 13, No. 2, Apr. 1995, Ipswich GB, pp. 30–42.

"Distributed Control Node Architecture in the Advanced Intelligent Network", M. Hirano et al, XV International Switching Symposium, Berlin, Apr. 23, 1995, pp. 278–282.

"CORBA: A Common Touch for Distributed Applications", F. Tibbits, *Data Communications*, vol. 24, No. 7, May 21, 1995, pp. 71–75.

"The Information Services Supermarket—an Information Network Prototype", I. Marshall et al, *BT Technology Journal*, vol. 13, No. 2, Apr. 1995, Ipswich GB, pp. 132–142.

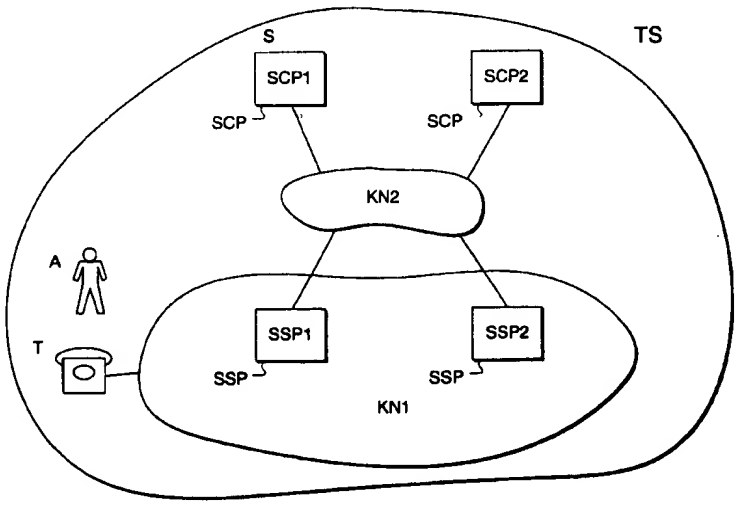
* cited by examiner

Primary Examiner—Creighton Smith
(74) *Attorney, Agent, or Firm*—Ware, Fressola, Van Der Sluys & Adolphson LLP

(57) **ABSTRACT**

The invention concerns a method for providing a service for users of a telecommunication network. Service calls requesting the execution of the service for an respective one of the users are routed to a service switching exchange of the telecommunication network or are respectively routed to one of several service switching exchanges of the telecommunication network. A corresponding service request is sent by the respective service switching exchange, that has received the respective service call, to a service control facility or to one of several possible service control facilities. The service request is routed to a particular one of a plurality of servers of the service control facility, which are formed on top of an object infrastructure within an object oriented computing environment The particular server acts as service repository server and determines another one of the plurality of servers that is available and able to execute the control of the service for the respective service call.

22 Claims, 5 Drawing Sheets



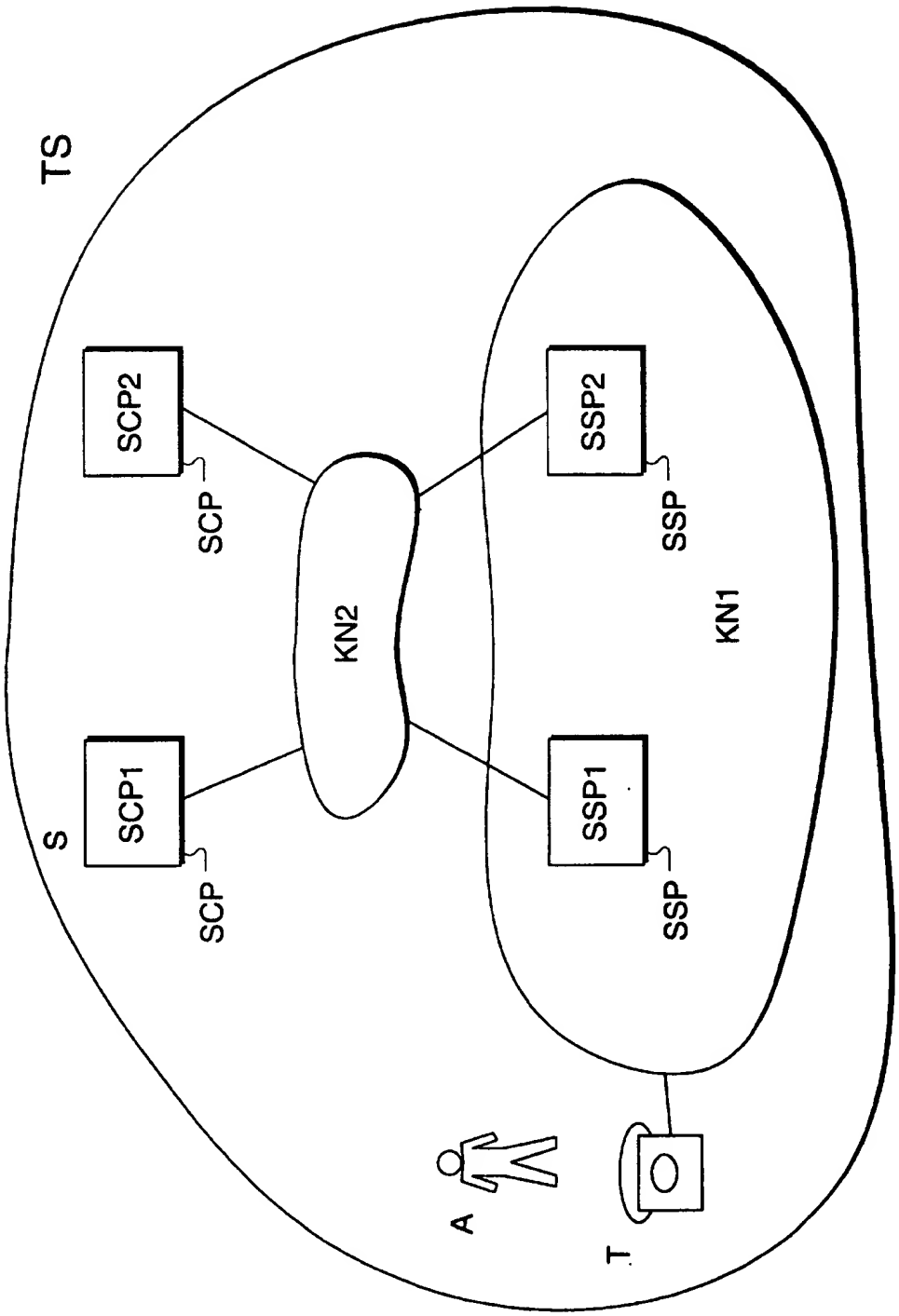


FIG. 1

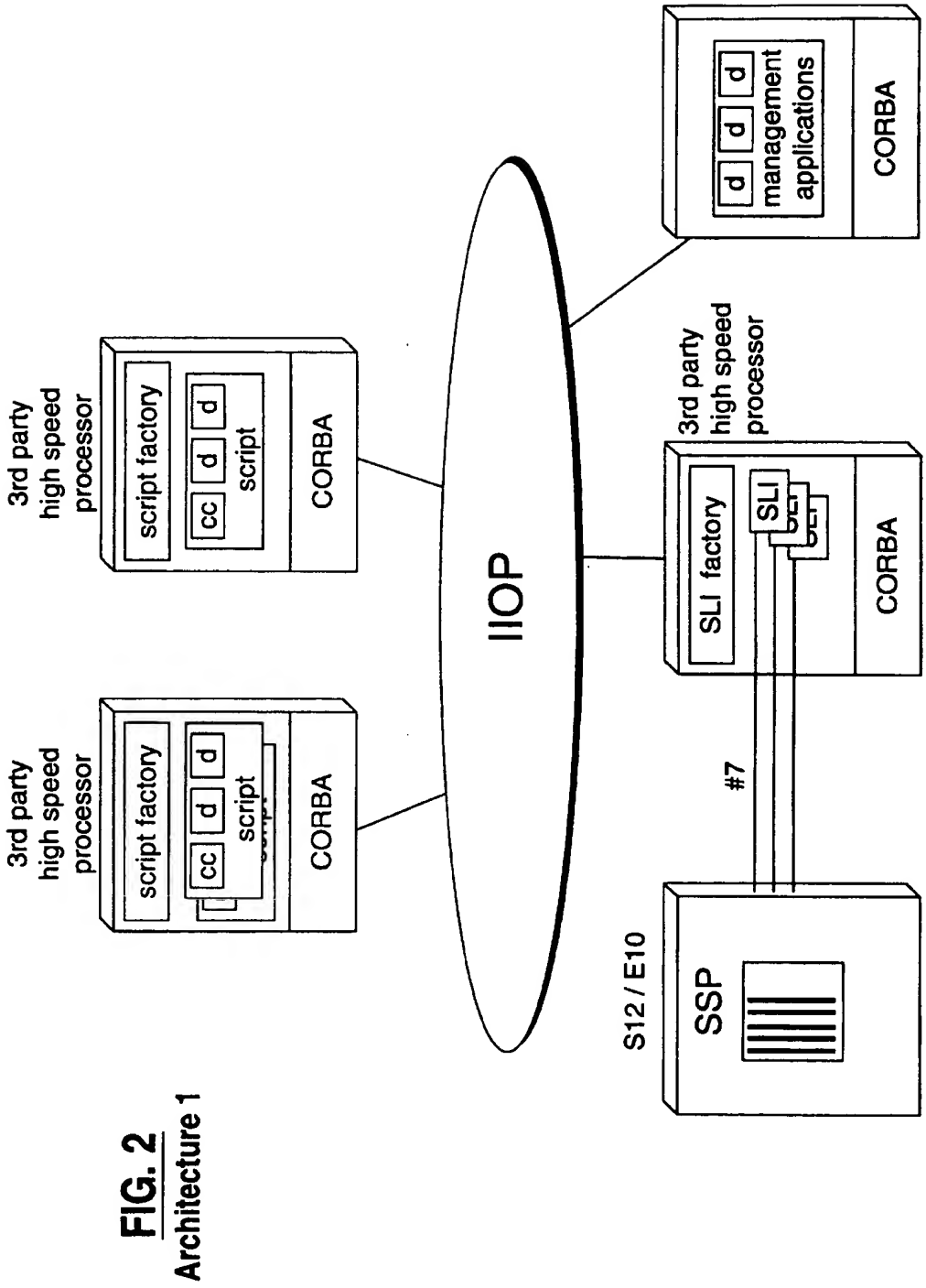


FIG. 2
Architecture 1

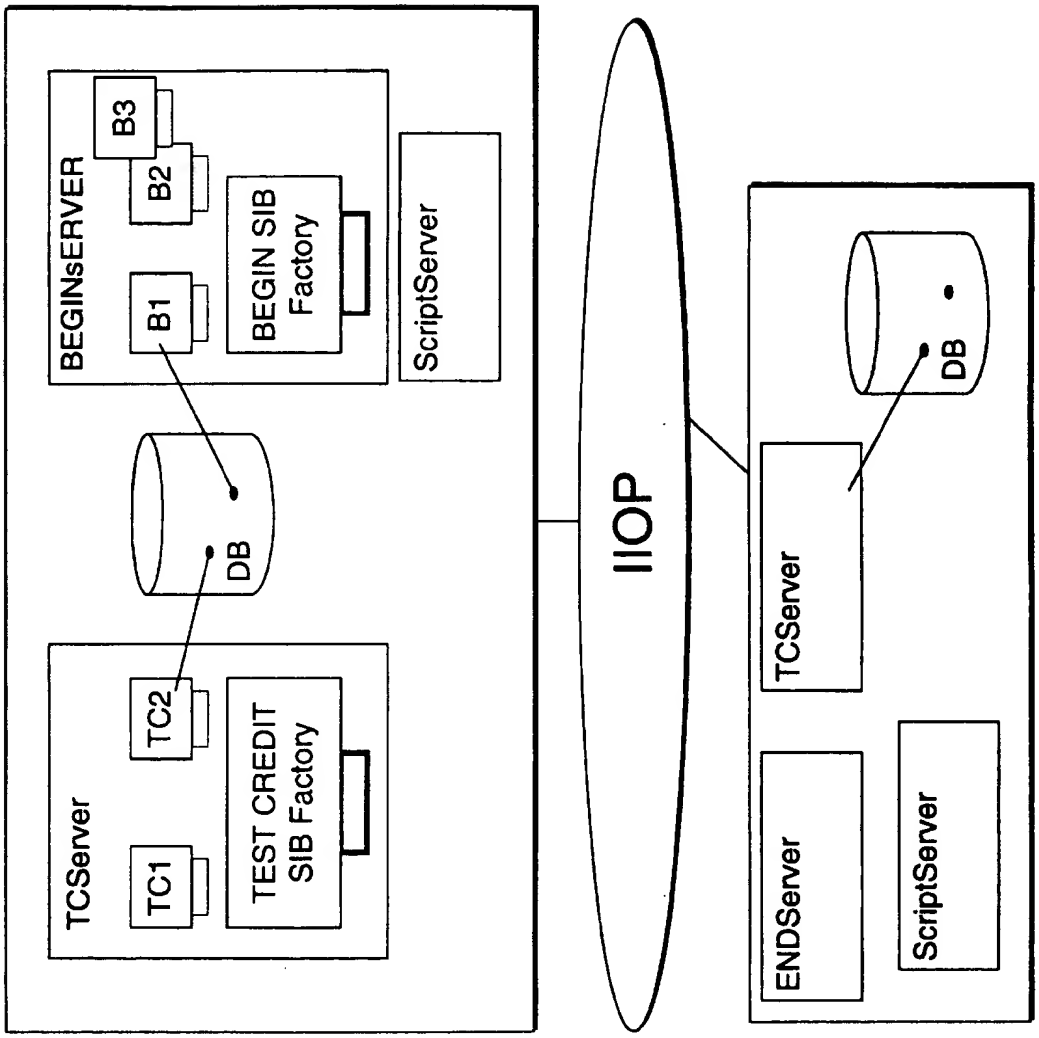


FIG. 3
Architecture 2

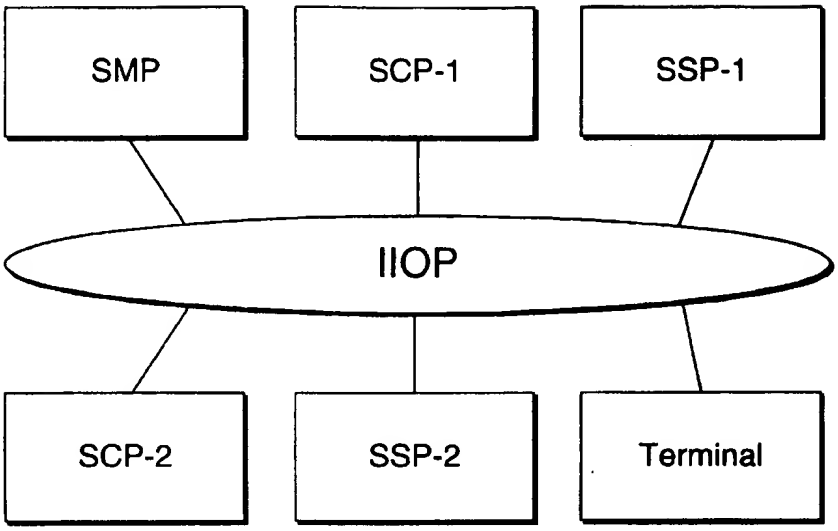


FIG. 4
Architecture 3

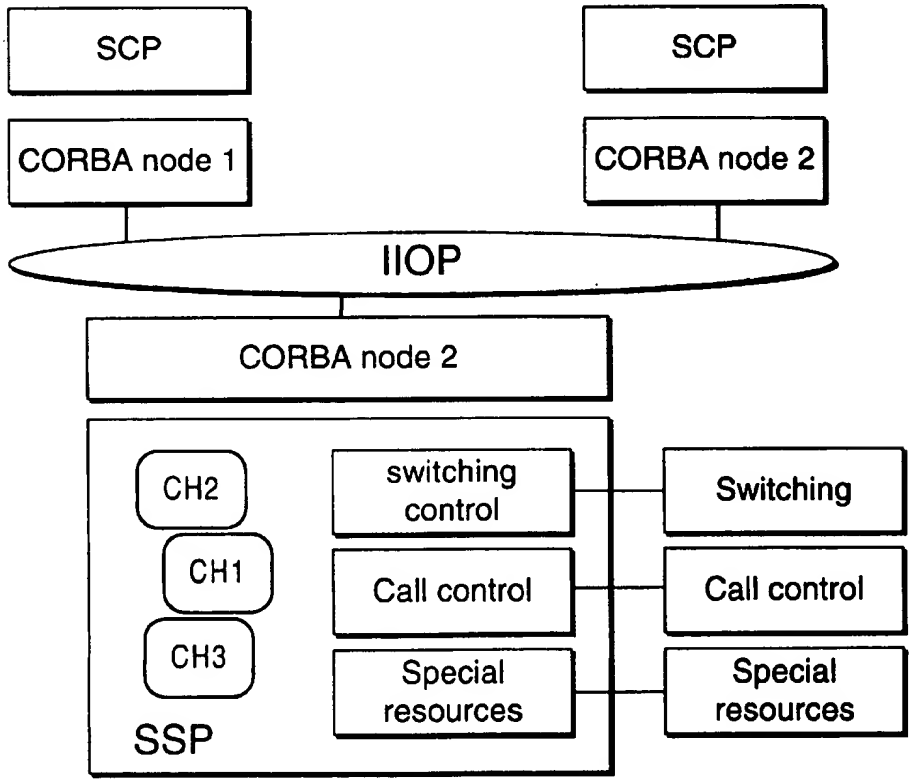


FIG. 5
SSP Server

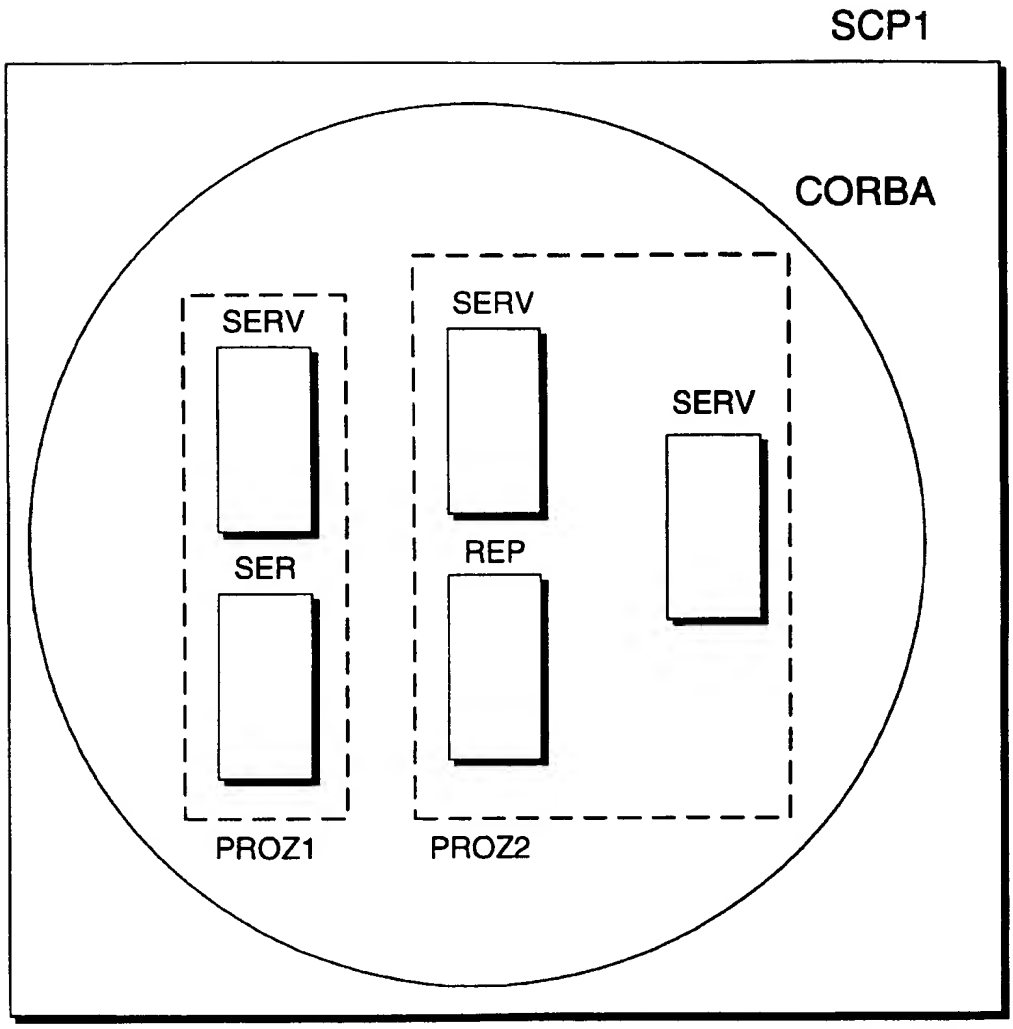


FIG. 6

METHOD FOR PROVIDING AT LEAST ONE SERVICE TO USERS OF A TELECOMMUNICATION NETWORK, SERVICE CONTROL FACILITY AND SERVER NODE

CROSS-REFERENCE TO RELATED APPLICATION

This application discloses subject matter that is disclosed and which may be claimed in copending applications entitled, "Method for Communicating Between a Service Switching Exchange of a Telecommunication Network and a Service Control Facility", filed Apr. 9, 1998 (Atty. Docket 902-679); and "Method of Providing a Service to Users of a Telecommunication Network, Service Control Facility, and Processing Node", filed on even date herewith (Atty. Docket 902-682); both of which are hereby incorporated by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

The invention concerns a method for providing at least one service to users of a telecommunication network, a method for provisioning the execution of service logic functions within a service control facility of a telecommunication systems, a service control facility for connecting to one or several service switching exchanges of a telecommunication network and a server node for a service control facility connected to one or several service switching exchanges of a telecommunication network.

2. Discussion of Related Art

Telecommunication services can be provided according to the Intelligent Network Architecture.

A user of a telecommunication network requests a service by dialing the number dedicated to the service. The call with this number is routed through the telecommunication network to a service switching exchange which recognizes this call as a call requesting the service. Then, this service switching exchange communicates via a data network with a service control facility, the so called service control point. This service control point contains a service logic which contains a description of the method that has to be executed to provide the service to the calling user. By a request of the service switching exchange the execution of this method by the service logic is initiated and the requested service is provided to the requesting user.

The service switching exchange is closely linked to a service control facility. The configuration of this link is made through basic proprietary tools provided by the service switching exchange/service control facility pair vendor and do not allow easy and dynamic configurations.

This service logic is realized as a single process handling all calls one after the other. Each of them going through a single queue. Each call is associated with a context allowing the state of the call not to be lost between user interaction. The service is executed through a finite state machine, taking as input the TCAP message and the context of the call.

The disadvantage of this approach is that this architecture of provisioning services is inflexible and does not make good use of the underlying data processing systems.

SUMMARY OF INVENTION

Accordingly, it is a primary objective of the present invention to improve the data processing characteristic of a service control facility.

A first aspect of the invention is a method for providing at least one service to users of a telecommunication network, in which method service calls requesting the execution of the service for an respective one of the users are routed to a service switching exchange of the telecommunication network or are respectively routed to one of several service switching exchanges of the telecommunication network. Also in this method of providing service, a corresponding service request is sent by the respective service switching exchange, that has received the respective service call, to a service control facility. This method is characterized in that the service request is routed to a particular one of a plurality of servers of the service control facility, which are formed on top of an object infrastructure within an object oriented computing environment. This method is also characterized in that the particular server acts as service repository server and determines another one of the plurality of servers that is available and able to execute the control of the service for the respective service call.

A second aspect of the invention is a method for provisioning the execution of service logic functions within a service control facility of a telecommunication system, where service requests, that request the execution of a service logic function, are received by the service control facility from at least one service switching exchange of the telecommunication system. This method is characterized in that the service request is routed to a particular one of a plurality of servers of the service control facility, which are formed on top of an object infrastructure within an object oriented computing environment. This method is also characterized in that the particular server acts as service repository server and determines another one of the plurality of servers that is available and able to execute the respective service request.

A third aspect of the invention is a service control facility for connecting to one or several service switching exchanges of a telecommunication network, where the service control facility contains means for receiving service requests, requesting execution of a service for a user of the telecommunication network from at least one of the service switching exchanges of the telecommunication network. This service control facility is characterized by containing a plurality of servers, which are formed on top of an object infrastructure within an object oriented computing environment, by containing means for routing the service request to a particular one of the service control facility, and by containing the particular server acting as service repository server and determining another one of the plurality of servers that is available and able to execute the respective service request.

A fourth aspect of the invention is a server node for a service control facility connected to one or several service switching exchanges of a telecommunication network, where the server node contains means for receiving service requests, requesting execution of a service for a user of the telecommunication network from at least one of the service switching exchanges of the telecommunication network. This server node is characterized in that the server node is formed on top of an object infrastructure within an object oriented computing environment, and contains means for determining another one of a plurality of server nodes which are formed on top of the object infrastructure within the object oriented computing environment. This server node is also characterized in that it is available and able to execute the respective service request.

The underlying idea of this invention is to implement each service that is offered by the service control facility by one

or many servers, which are formed on top of an object infrastructure within an object oriented computing environment and to provide a particular server, a service repository server which determines the one of the servers that has to execute the service for an incoming service request.

Therefore, the service architecture is no longer based on a functional design and technologies like multithreading or multi processing can be applied. Further, an open way to distribute service execution united around one or several service switching exchanges is offered.

With the service repository server distribution is transparent and system configuration is very easily achieved through plug and play. The scalability of the system is ensured and easy to realize through an open interface.

The introduction of CORBA based distribution provides high scalability and IT platform independence. It allows for easy service interworking personalization, distribution and maintainability of the software.

In the approach where the link between the service control facility and the service switching exchange conforms to a OMG CORBA 2.0 specification, the configuration of processing nodes and of server nodes must take advantage of the underlying CORBA platform.

Due to the introduction of an object oriented computing environment and the special object design described above, the processing of the service requests can be distributed and this proved high scalability and IT platform independence. The redesign of the service logic according to this approach allows for easy service interworking personalization, distribution and maintainability of the software. The design pattern allows for taking all benefits from object oriented programming.

Furthermore, it allows introduction of multithreading. The direct benefit is that a service session, that means the processing of a service request, is not blocked by the execution of another one.

The use of a factory allows implementation of several life cycle policies for the service session objects.

It is further advantageous

- to manage the creation of a service session object by a service dedicated factory.
- to set the interface definition of a service session object as TCAP mapped over IDL. (Interface Definition Language which is used to describe the interface of an object in language-neutral terms).

By the use of CORBA servers the service control facility implementation is simplified and scalability is ensured.

These and other objects, features and advantages of the present invention will become more apparent in light of the detailed description of a best mode embodiment thereof, as illustrated in the accompanying drawing.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram presenting the architecture of a telecommunication system containing a service control facility according to the invention.

FIG. 2 is a block diagram presenting a first object architecture of the telecommunication system.

FIG. 3 is a block diagram presenting a second object architecture of the telecommunication system.

FIG. 4 is a block diagram presenting a third object architecture of the telecommunication system.

FIG. 5 is a block diagram presenting a object architecture of a service switching exchange for the telecommunication system.

FIG. 6 shows the service control facility SCP1 with several servers SERV and REP that are formed on top of an object infrastructure CORBA ORB in an object oriented environment CORBA.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIG. 1 shows a telecommunication system TS with an telecommunication network KN1, a subscriber terminal T assigned to a subscriber A, a communication network KN2 and two service control facilities SCP1 and SCP2.

The telecommunication network KN1 is for example a telephone network. It is also possible that this network is a data network or a communication network for mixed data speech and video transmission.

From the exchanges of the network KN1 two special designed exchanges SSP1 and SSP2 are shown. These exchanges are service switching exchanges that contain a service switching functionality SSP. To these exchanges service request calls from the subscriber of the network are routed. When such a service switching exchange receives such a service request, it sends a corresponding request via the network KN2 to one of the two service control facilities SCP1, SCP2. The service is then provided under the control of the respective one of the service control facilities SCP1 and SCP2.

Each of the service control facilities SCP1 and SCP2 provides one or several services S and realize a service control function.

The communication network KN2 is preferably a data network, especially according to the SS7 signaling protocol. It is also possible that this network is a LAN (Local Area Network), a MAN (Metropolitan Area Network) or an ATM (Asynchronous Transfer Mode) network.

Preferably, the service switching exchanges SSP1, SSP2, and the communication between them and the service control facilities SCP1, SCP2 are designed according to the Intelligent Network Architecture (according to the service switching point and according to the communication between the service switching point and the service control point), described for example in the ITU-T Q.1214 Recommendation.

If a user wants to start an service S, he dials a specific number for that service. The local exchange will route this call to the nearest service switching exchanges and this will trigger the service switching functionality SSP to start the service. This is done by sending a request to the SCP to start executing a Service Logic Program (SLP). An SLP is specific for each service, it will instruct the network on how to handle the service. It will execute service logic, statistic logic, database logic, charging logic, etc. It will also provide instructions for the switch (e.g., create the second leg of the call) and Intelligent Peripherals (IP) (e.g., announcement machines). A protocol according to the Information Networking Architecture (INAP) protocol is used for this, and INAP messages are transported in TCAP messages between the SCP and SSP.

When designing a distributed application, first the possible system components to be distributed are identified (i.e., what the objects of the system are and how they will be grouped into larger objects that will be distributed).

The breaking down of the SCP into objects offers some possibilities. FIGS. 2 to 3 depict these organized into different architectures. All state information for handling one call was kept together and that all needed data was organized

into objects. So, one possibility for SCP objects is a script object that executes the service logic for one call and an object per data object.

The values of the data objects are kept in a database, so having data objects that hide the use of a database is preferable (i.e., certain independence on the database). The catch lies in the fact that making these objects CORBA objects gives us a considerable overhead when methods (e.g., to read or write attribute values) are called on them. So, the system performance would become unacceptable.

The script object could be decomposed into a service execution engine and an SIB graph. So, SIBs could become distributed objects too. If database objects are distributed, this offers the possibility to execute an SIB on the machine where the particular data object resides. We can envisage a scheme where there is a control entity (a script) that executes SIBs remotely (possibility of having more overhead) or a scheme where control is passed from SIB to SIB. This trade off between getting the data and running a script on one machine and running parts of a script (SIBs) where the data resides is certainly worth investigating.

The SSP's main functionality is to provide a hook between the call processing and the IN system. An SSP server must at least have an object that bridges between the normal Call Control Function (CCF) and an object that bridges between the Switching Function (SF). Also an object that bridges between Specialized Resources (SRF) (e.g., announcement equipment), seems to be obvious. In FIG. 5, these objects are depicted as a Switching Control, Call Control and Special Resources Object. We also need an object, Call Handler (CH) that interlaces with the SCP and the other objects in the SSP. There are two possibilities: there will be only one that handles all calls, or several, each handling one call. If we have several, existing only for the duration of the call, we also need a corresponding factory object to control the lifecycle of these objects.

In the following is given the architecture of an SSP server on CORBA. For the Switching Control, Call Control and Special Resources objects we could have two approaches. In the first approach they are only interface objects between the proprietary Application Program Interface (API) of existing software. In the second approach they would control the hardware directly, what implies providing a CORBA interface to Switching Control, Call Control & Special Resources. In theory this would be the best solution, since the approach would give an overall consistent software architecture, in this case special IDL (Interface Definition Language) adapter interfaces have to be designed.

A SMP (Simple Management Protocol) is responsible for service control and monitoring. In order to be able to control services, the SMP does not need to be a DPE object. What is needed is that the objects that are controlled (e.g., script) provide an interface to do so. For monitoring we need an SMP object that is able to receive 'events' from the objects that are being monitored. In our research prototype we defined and implemented a trace control as an example on what is possible as management functionality on the SCP.

In the previous section we have described a number of possibilities for CORBA objects in an IN Architecture. In this section we will group certain of these CORBA objects into a specific architecture. The sequence of the architectures can be seen as a possible evolution scenario, starting from current day's service provisioning, especially IN, products.

The basis of the first architecture is the usage of a distributed database for data distribution and CORBA distribution for distributed SCP. This architecture is designed to interface with existing SSPs, so the SSP is not a CORBA object. Since this architecture was used to implement a research prototype, we elaborate on it more than the others. FIG. 2 depicts the architecture.

The basic components in the architecture are:

the script: a CORBA object that handles exactly one call

the SLI (Service Logic Interface), a CORBA object that handles a TCAP dialogue for, and interfaces with the SSP.

the distributed DB.

Existing IN service scripts can encapsulated in Script objects.

A Script object also contains the Service Data and Call Context related to a particular call, and one Script object is instantiated per call. The interface of the Script object contains operations that represent the TCAP messages that the Script receives during its execution.

A special Service Logic Interface (SLI) object receives the #7 messages from the SSP and dispatches them to the corresponding script object instantiation, using the ORB services. The interface of the SLI object contains the TCAP messages that can be sent to it. One SLI object is instantiated per call.

Since the Script and SLI objects exist only during the lifetime of a call corresponding factory objects are used to create and destroy them. Data objects are not CORBA objects, they are implemented as C++ objects. This approach hides the use of a particular database from the service logic.

It is possible to use as database for example Oracle, SQLNET or ObjectStore. The SMP has not be considered; we could have used the proprietary mechanism to send status information to it, or defined it as a CORBA object with an appropriate IDL interface.

However as an example on what is possible, its easy possible to design and implement a small trace control system; each server has a tracer object, the trace control part of the SMP can call methods to turn traces on particular implementation objects on and off. The SMP also manages the database (using SQLNET).

A script and a SLI are CORBA objects, how they are distributed over servers is a deployment issue. A CORBA server is a process running at a particular node in the system. There is one SLIServer in the system that interfaces with the SSP. It has one CORBA object, the SLIFactory, that is always present when the server is running.

The main purpose of this object is to create new SLI objects when new calls are started (when a TCAP dialogue is opened). The SLIServer is deployed in the node that has the #7 hardware and software installed. The system can have a number of ScriptServers, one per available node.

This server also has a factory object, that exists during the lifetime of the server itself. This factory object is registered in the CORBA Naming service, so that an SLI can find it. In this way service logic distribution is facilitated. Adding more processing power to the system just means installing the new machine, running the ScriptServer on it, and the next time the SLI looks for a ScriptFactory a new possibility is available. The same mechanism also provides some reliability, when a machine goes down for some reason, the SLI will simply not get a reference to that ScriptServer any more. Note that the ScriptFactory is also to best candidate to put a management interface on (e.g., to change the service characteristics), since it is always available as long as the server is running.

The interface between SLI and Script is defined to be asynchronous and in terms of TCAP primitives (BEGIN, INVOKE, RESULT, . . .). This option is advantageous, since

one of the requirements is to reuse existing IN product software. Another option is to define this interface in terms of INAP primitives.

The difference between a CORBA object, a (CORBA) server and a process must be clear. A CORBA object implements an interface and runs within a CORBA server. The CORBA server itself is one process that runs on a host (or node). This definition is important for the relation between a script and a call. In the current ALCATEL IN product implementation a script is a process that handles different calls, and keeps a different call context for each of those. This is done for performance reasons, starting up a separate process (script) for each call takes some time.

We could adapt a similar approach (one script handles n calls) in these architectures, but the fact that a script has multiple contexts is not that clean and it is possible to deal with the performance issue by having a CORBA server with multiple scripts (each handling one call). So, it is better that in this architecture, a host can run one or more CORBA servers, each running different script objects that each handle a specific call.

The CORBA server is similar to the script in the current IN product, but the problem of handling multiple contexts is shifted from the application programmer to the CORBA platform.

A step further in this sequence of architectures, as second architecture, is to break down the script object into its components. As known, IN services are designed as a number SIBs that are nodes in a service logic graph. The execution of the script means executing SIBs and moving to the next SIB depending on data and responses coming from the SSP. Instead of designing this as a monolithic piece of software (from the SCE), it could be designed in terms cooperating SIBS. As known, the IN capability set 1. SIBs are not sufficiently defined to used them as they are. This resulted in IN platform providers having their own set of SIBs derived from the standard and software from SIBs is not reusable among platforms.

When SIBs are defined as CORBA objects this situation could be broken and service providers would be able to design their own services, reusing existing SIBs (source code).

Having SIBs as independent CORBA objects offers the possibility to execute the SIBs in different places. Instead of getting the data (using a distributed database, or using CORBA objects) to the place where the script or SIB is executed, we could start the SIB on the machine where the data is located. This brings benefits. We can envisage a scheme where there is a control entity (a script) that executes SIBs remotely or a scheme where control is passed from SIB to SIB.

A third architecture breaks with traditional IN product in the sense that it also makes the SSP a CORBA object. In the previous architectures, for reliability reasons, it is still necessary that CORBA servers are deployed on machines that are interconnected on a reliable LAN. In practice, with the current DPE implementations (TCP/IP) this means putting the machines in the same location. This technology restriction prevents putting the SSP on the (Distributed Processing Environment), since in practice SCP and SSP sites are located on different places. However, technology evolves and DPE providers (e.g., IONA) are looking into the possibility of porting their product to other transport protocols, e.g., #7 links for telecommunication purposes. When this happens the step to making the SSP a CORBA object is not that big. In this case, the SLI object would not be needed anymore, since the SSP object could take over its role. When a call is started it gets a reference to a Script-Factory and asks to create the script object itself. It could then communicate with this object directly. Also the language used in this communication (INAP encapsulated in a TCAP based interface) can be simplified. This interface could be defined in terms of INAP primitives (e.g., provide_instruction(args), join(args)). This would improve (simplify) the system design and implementation, since the TCAP layer disappears for the application programmer. One could say that when CORBA is used, the application programmer must only be concerned with the application layer of the communication, not with the session layer (when TCAP is used).

The final step would be to extend integrate a terminal (as proposed by TINA (Telecommunications Information Networking Architecture)). This would mean a major change in user equipment. However this change is not infeasible if we consider the fact that more and more users get connected to the internet over their telephone lines. And the technology needed to have the DPE extended to the terminal is similar to this. The benefit of this would be that the terminal to network protocol could become much richer than just passing DTMF tones as numbers.

The above described architecture provides a solution to make especially IN platforms more scalable. Since interworking with and evolution from existing products is important we presented a sequence of architectures that can be seen as a possible evolution scenario.

FIG. **6** shows the service control facility SCP1 with several servers SERV and REP that are formed on top of an object infrastructure CORBA ORB in an object oriented environment CORBA.

The different servers runs on different processing nodes PROZ1 and PROZ2.

Each service that is provided by the SCP1 is implemented by one or many servers SERV. Each call to the service is a single CORBA object managed by one of the corresponding servers SERV through a factory. SSP1 and SSP2 has access to several servers SERV on top of the CORBA ORB. That is, as if they have access to several service control function SCPs. To provide the link a service repository server REP is designed.

Upon request for a new service session, SSP1 accesses the service repository server REP to find an available service server SERV to execute the service session.

Input information to the service repository server is made of the IN service name and a service session key. The matching object reference of the service factory is returned by the service repository server.

A service server is registered into the service repository based on a constraint on the session key. Thus, a key definition language has been defined in offer to process the key definition when registering a new server, and process the key when the SSP performs a query.

As an example an implementation procedure according to the architecture **3** is described in the following:

A Credit Card Calling service which existed for the UNIX based IN Release existing Alcatel product is to be implemented. The Credit Card Call service is a "typical" IN service and allows a call from any terminal to be billed to a particular card number. To set up that call, the user has to enter a card number, a PIN code and the destination number. If the entered data is correct, the call is completed to the destination address and charged to the user's card.

An SSP simulation stub was used to generate the TCAP dialogue for the Credit Card Call. The SSP stub is a CORBA object that sends and receives the same TCAP messages that

an SSP would. The SSP stub is capable of simulating different call scenarios, can generate different call loads and can repeat a scenario an arbitrary number of times.

Another general issue for the use of CORBA in an IN oriented service provisioning architecture is the interworking of CORBA-based IN with legacy IN applications.

This interworking can be achieved with the use of Adaptation. It is related to the way a CORBA based infrastructure can communicate with the legacy of switching systems, namely SSPs.

There are two approaches possible:

1) Definition of an Adaptation Unit which is an application level bridge that provide the mapping of SS7 and TCAP primitives into IDL invocations. This approach is similar to one taken by the XoJIDM group (Joint Inter Domain Task of X/Open) for the CORBA/CMIP gateway. The result of their works can be reused especially the static mapping of GDMO/ASN.1 to IDL interfaces (GDMO is an acronym for "Guidelines for the Definition of Managed Objects," and ASN.1 stands for "Abstract Syntax Notation One").

   In order to minimize the impact on existing systems, CORBA should provide framework services and tools in order to achieve this mapping. The availability of such framework will allow interworking of CORBA-based IN with a variety of existing hardware such as SCPs and HLRs.

   Such application level bridge should be characterized by high availability and high performance without being a bottleneck for a distributed system. However for the required real-time performance, this is an intermediate solution towards full IN CORBA-based systems.

   This approach would involve building an IDL interface to represent application level protocols such as MAP and INAP and others which are based on the TCAP layer. The TCAP layer provides a 'ROSE-like' asynchronous RPC service over the SCCP layer. And basing the bridge on TCAP will exclude the use of circuit related signaling protocols such as ISUP and TUP from the solution.

   Like in the XoJIDM approach, there should be a static translation of the INAP/TCAP specification to IDL interfaces which will be done by a dedicated compiler. Thus, any INAP dialect can be converted to appropriate IDL interfaces. These applications level bridges would implement these IDL generated interfaces and dynamically perform conversion between IDL-derived types and their ASN.1 equivalents.

   CORBA nodes using the protocol specific bridges would generally run two servers, one each for processing outgoing and incoming invocations. Since TCAP is a connectionless service the gateway will have to maintain state in order to match invocations to responses and exceptions.

2) Usage of SS7 as an Environment Specific Inter-ORB protocol (ESIOP):

   A possible solution is to map the CORBA GIOP (General Inter-ORB Protocol which makes requests or returns replies between Object Request Brokers) on top of SS7 or to build a new Extended protocol (ESIOP) on top of the SS7 layers.

   The ESIOP solution would essentially use an SS7 protocol-as a transport for the GIOP protocol. CORBA objects would be visible across the SS7 network in exactly the same manner as they would be across the Internet with the CORBA IIOP. This would allow as ORB to use existing SS7 infrastructure as transport.

It should be noticed that existing signaling networks were never dimensioned to support the sort of traffic which will be exchanged by CORBA nodes. However, there is a potential benefit in this approach by exploiting the fault tolerant nature of the SS7 network.

The first issue here is the choice of SS7 protocol access level for this solution. The two principal choices are TCAP and SCCP:

GIOP at TCAP level:

It should be possible to use TCAP for carrying GIOP messages since IT is essentially a ROSE-like service. Services which make use of TCAP must define their operations using ASN.1 modules. It is envisaged that ASN.1 macros would be used to define the operation set: Request, Reply, CancelRequest, LocateRequest, LocateReply, CloseConnection and MessageError. These operations correspond to the GIOP primitives. The gateway would be responsible for converting CDR format, of the form used by GIOP, into a form transportable using ASN.1 types (possibly as an opaque buffer with BER encoding).

While it should be possible in principle to use TCAP as the basis for the ESIOP, it is not suitable because of:

the complexity of the implementation.

the overhead incurred in by the TCAP layer in addition to basic transport

the asynchronous nature of the protocol.

ESIOP at SCCP level:

The other choice for implementing the SS7 ESIOP is at the SCCP level. The SCCP provides services corresponding to the OSI-RM network layer functionality. It can operate in both connection-oriented and connectionless mode. It should be feasible to use the SCCP to transport GIOP messages although the ESIOP code may be required to perform its own transport layer functions (e.g., end-to-end flow-control and segmentation/re-assembly). It should be possible to address CORBA nodes using the "Global Title" mode of SCCP addressing. It would appear that using SCCP as the basis for an ESIOP for the SS7 network would be the best approach.

The following requirements are advantageous for an object oriented environment for an infrastructure in which a service session object interacts (described by hand of the CORBA environment):

In terms of functional requirements, CORBA should provide:

asynchronous Invocation model and support for group communication in the ORB;

proper hooks for extending the ORB with security, transaction and replication mechanisms;

node management and minimal object lifecycle facilities;

a time service and native monitoring facilities for capturing both computational and engineering events;

support for multithreading with flexible event-to-thread mappings.

The non-functional requirements of CORBA are concerned with

identifying the level of salability and object granularity that the ORB should meet in the IN context;

identifying the performance level that the ORB must achieve in terms of latency, throughput, etc.;

providing a predictable ORB core by instrumenting and documenting the time behavior of the platform;

reliability which denotes requirements that define acceptable behaviors for the distributed applications in the presence of hardware and software failures. In this case

two types of reliability can be identified; the integrity state and the availability;

manageability which denotes requirements that enable ease of development, deployment and operation for complex applications. In this case maintainability, (re) configurability and extensibility of CORBA applications.

An asynchronous Invocation model commonly used by IN applications. Thus, asynchronous and optionally isochronous invocation model is required with the ORB. The requirement here is a client side programming model which potentially supports a lightweight asynchronous communication mechanism incorporating a mandatory callback (or ORB "upcall" type mechanism) and optionally a "Futures" type programming model.

For the same object, both asynchronous and synchronous models should co-exist. The maximum concurrency level is defined in the configuration (e.g., QoS parameters), and has to be controllable, with the possibility of queuing the request or returning an exception.

A basic fault detection support should be provided by the ORB runtime for any object which needs it. This can be done by a timer which is implicit set and managed in the client process.

Flexibility and scalability:

The ORB should be able to support applications handling a large number of objects and should be able to support many simultaneous connections to remote objects.

The memory cost of an unused remote interface reference in a given capsule (i.e., Unix process) should be of the order of a standard language pointer.

A typical telecommunications environment comprises objects of very different granularities in both space (memory size) and time (object lifetime and duration). A scalable ORB should support objects at different levels of granularity, and minimize the overhead associated with the handling of small and volatile objects and of connections to remote objects.

To achieve some of the ORB fault-tolerance and reliability, the CORBA object model could be enhanced to support groups of objects or globally known as group communication such as those described

Reliability and availability:

To achieve the reliability requirements, the notion of replicated distributed objects can be introduced in CORBA. Thus the critical components of the application are implemented through replicated objects. This replica management can be handled automatically by the ORB such that clients interacting with a given server object is not aware if it is replicated or not. The degree of replication can be determined by the desired level of reliability which can be measured with the number of maximum number of concurrent failures; the nature of the failure which can be typed (malicious or benign); and the type of reliability property being sought such as integrity or availability. High availability or Fault Tolerance capabilities should be provided by CORBA for distributed IN systems in order to ensure the same robustness as current IN systems.

Timeliness requirements can be also achieved by the replica service. For example, applications that need to have bounded and predictable delays will be implemented through replicated objects. Each replica is able to process locally all of the methods defined for the object. In the absence of failure, a client can use any of the responses to a method Invocation as its result, (the invocations in this case is perform synchronously).

To achieve this, the ORB will have a modular structure allowing the implementation of different profiles, depending on application or services requirements. These profiles give an abstraction of the resources provided by the underlying operating system. One of the ORB module will be a failure detector which is at the lower level of the ORB structure. And a response Invocation delay beyond a certain threshold are classified as failure. Thus, the client is able to trade off reliability for timeliness, the shorter the threshold, the tighter the bounds on delays. By replicating an object in a sufficient number of times, the client is able to meet both timeliness and reliability requirements simultaneously.

Here, we have identified a requirement for a replication service with correctness, safety and liveness properties; and a failure detector module which is part of the ORB core.

Performance:

Performance of IN are measured in number of calls per second for service nodes and intelligent peripherals; and number of transaction per second for signaling control point. These calls and transactions may involve multiple messages exchanged between an SSP and the Intelligent Layer.

To obtain the actual performance of legacy IN systems, real-time performance of the ORB is required for the use of distributed processing in IN systems as well as its distribution on a geographical scale (non centralized IN systems). To achieve better performance for IN distributed systems, the ORB call overhead should be reduced, and the performance level that the ORB must achieve in terms of latency, throughput, etc. should be defined and documented.

What is claimed is:

1. A method for providing at least one service to users of a telecommunication network, in which method service calls requesting the execution of the service for an respective one of the users are routed to a service switching exchange of the telecommunication network or are respectively routed to one of several service switching exchanges of the telecommunication network and in which method a corresponding service request is sent by the respective service switching exchange, that has received the respective service call, to a service control facility, characterized in that the service request is routed to a particular one of a plurality of servers of the service control facility, which are formed on top of an object infrastructure within an object oriented computing environment, that the particular server acts as service repository server and determines another one of the plurality of servers that is available and able to execute the control of the service for the respective service call.

2. A method as claimed in claim 1, characterized in that different services are provided by the service control facility and that a service logic function for each of these different services is implemented within one or several of the servers.

3. A method as claimed in claim 1, characterized in that the particular server determines an object reference of a service factory object that creates a service session object which is able to control the execution of the service for the respective service call, said factory object being an object which provides the function of creating a specific kind of object.

4. A method as claimed in claim 1, characterized in that a service session object, which manages a service session, is managed by one of the servers through a factory object.

5. A method as claimed in claim 1, characterized in that for each service request one service session object, which is able to interact and communicate via an object infrastructure with other objects in a object oriented computing environment, is created and the respective service session object controls the execution of the service for the respective service call.

6. A method as claimed in claim 1, characterized in that the particular server receives as input data a name and a

service session key and returns a matching object reference of a service factory object, wherein said session key is a key used for just one message or set of messages.

7. A method as claimed in claim 1, characterized in that the other servers are registered into the particular server based on a constraint of a session key.

8. A method as claimed in claim 1, characterized in that the particular server performs load balancing between the servers that implementing the same service.

9. A method as claimed in claim 1, characterized in that the or each service switching exchange of the telecommunication network communicates with the or each service control facility according to the communication protocols of the Intelligent Network Architecture.

10. A method as claimed in claim 1, characterized in that the plurality of servers is formed on top of CORBA object infrastructure.

11. A method as claimed in claim 1, characterized in that the creation of each service session object is managed through a factory object.

12. A method as claimed in claim 7, characterized in that the factory object implements a life cycle policy for the service session object, wherein the factory object creates the service session object and destroys the service session object when the service session object's life cycle is over.

13. A method as claimed in claim 7, characterized in that the factory object implements a creation on demand life cycle policy for the service session object.

14. A method as claimed in claim 7, characterized in that the factory object implements activation on demand life cycle policy for the service session object.

15. A method as claimed in claim 1, characterized in that the service session object has, as its interface definition, TCAP mapped over IDL.

16. A method as claimed in claim 1, characterized in that the service session object receives directly its own message invocations.

17. A method as claimed in claim 1, characterized in that the service session object receives INAP messages from the or each service switching exchange as message invocations.

18. A method as claimed in claim 1, characterized in that each service session object runs in its own thread.

19. A method for provisioning the execution of service logic functions within a service control facility of a telecommunication systems, where service requests, that request the execution of a service logic function, are received by the service control facility from at least one service switching exchange of the telecommunication system, characterized in that the service request is routed to a particular one of a plurality of servers of the service control facility, which are formed on top of an object infrastructure within an object oriented computing environment, that the particular server acts as service repository server and determines another one of the plurality of servers that is available and able to execute the respective service request.

20. A service control facility for connecting to one or several service switching exchanges of a telecommunication network, where the service control facility contains means for receiving service requests, that request the execution of a service for a user of the telecommunication network, from at least one of the service switching exchanges of the telecommunication network, characterized by containing a plurality of servers, which are formed on top of an object infrastructure within an object oriented computing environment, containing means for routing the service request to a particular one of the service control facility, and containing the particular server acting as service repository server and determining another one of the plurality of servers that is available and able to execute the respective service request.

21. The service control facility according to claim 16, characterized by containing a variable number of processing nodes processing the plurality of servers.

22. A server node for a service control facility connected to one or several service switching exchanges of a telecommunication network, where the server node contains means for receiving service requests, that request the execution of a service for a user of the telecommunication network, from at least one of the service switching exchanges of the telecommunication network, characterized by that the server node is formed on top of an object infrastructure within an object oriented computing environment, and containing means for determining another one of a plurality of server nodes which are formed on top of the object infrastructure within the object oriented computing environment, that is available and able to execute the respective service request.

* * * * *

US005717747A

United States Patent [19]

Boyle, III et al.

[11] Patent Number: 5,717,747

[45] Date of Patent: Feb. 10, 1998

[54] **ARRANGEMENT FOR FACILITATING PLUG-AND-PLAY CALL FEATURES**

[75] Inventors: **Frank J. Boyle, III**, Broomfield; **Andrew D. Franklin**; **Jane Gambill**, both of Boulder; **Charles H. Parker**, Broomfield; **Dennis R. Sanger**, Westminster, all of Colo.

[73] Assignee: **Lucent Technologies Inc.**, Murray Hill, N.J.

[21] Appl. No.: **656,517**

[22] Filed: **May 31, 1996**

[51] Int. Cl.[6] **H04M 3/42**; H04M 3/00

[52] U.S. Cl. **379/201**; 379/242; 379/269

[58] Field of Search 379/201, 207, 379/242, 268, 269, 280, 284

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,695,977	9/1987	Hansen et al.	364/900
4,747,127	5/1988	Hansen et al.	379/94
4,782,517	11/1988	Bernardis et al.	379/201
5,337,351	8/1994	Manabe et al.	379/201
5,404,396	4/1995	Brennan	379/207 X
5,448,631	9/1995	Cain	379/207 X
5,471,318	11/1995	Ahuja et al.	358/400

OTHER PUBLICATIONS

U. S. Patent Application, S. R. Ahuja 7-2-3-1-1-1-7, Serial No. 08/051724, "Multimedia Telecommunications Network and Service", Filed Apr. 22, 1993.

The AT&T MultiMedia Communications eXchange, Publication No. GBCS-MCS-097P1, 1996 (4 pages).

D. A. Berkley, et al., *Multimedia Research Platforms*, AT&T Technical Journal, Sep./Oct., vol. 74, No. 5, pp. 34-53.

J. O. Coplien, *Ishmael: An Integrated Software/Hardware Maintenance and Evolution Environment*, AT&T Technical Journal, Jan./Feb. 1991, vol. 70, No. 1, pp. 52-63.

Communique! InSoft$_R$ Desktop Conferencing and Workgroup Collaboration User's Guide, Version 4 for use with the Motif Window Manager, Apr. 6, 1995, pp. 9, 17-43.

InSoft Brochure: *InSoft^R Digital Video Everywhere, DVE*, (2 pages), 1994.

InSoft Brochure: *InSoft^R Open DVE^{sm}*. (2 pages), 1994.

InSoft Brochure: *Turn Your Desktop Into A Global Conference Room*, InSoft^R. (2 pages), (undated).

(List continued on next page.)

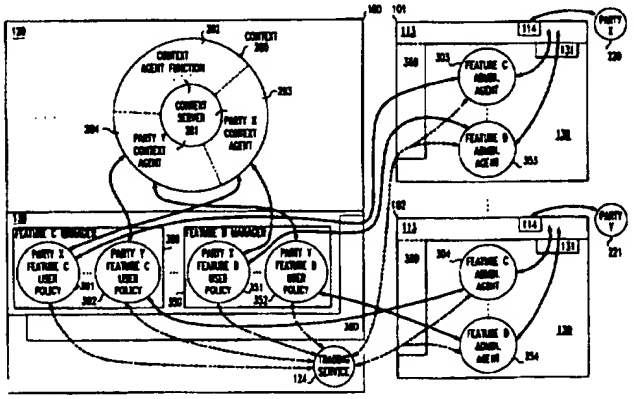
Primary Examiner—Harry S. Hong

Attorney, Agent, or Firm—David Volejnicek

[57] **ABSTRACT**

A telecommunications system infrastructure that facilitates easy insertion of feature software into existing such telecommunications systems and easy integration of the new calling features and their implementing software with existing features and their software. The infrastructure comprises the Lucent Technologies MMCX multimedia communications server (**100**) and middleware-compliant communications endpoints (**101–102**) executing the Lucent Technologies MMCX communications middleware (**111–112**). Feature-implementing software has a modular client/server construction, with feature managers (server modules, **300**, **350**) executing on the MMCX server and feature administration agents (client modules, **303–353**, **304–354**) executing on the endpoints. The infrastructure provides a context service (**120**) and a context API (**121**) for registering an instance of a feature manager (a user policy, **301–302**, **351–352**) for each user upon that user becoming entitled to the feature, an administration API (**360**) for communications between feature managers and feature administration agents on the user's endpoint to customize the user's user policies for the user, and a context (a cyberspace meeting room, **200**) and the context API for involving the user policies of users who are parties to a call in the call and for communicating call-related events to feature servers and other service-implementing software. Call-related events are passed to user policies involved in the call, and they are given a chance to react to the events by allowing or rejecting the events. Interactions between features are managed by having feature managers register at different priorities with the context service; higher-priority modules are serially given an opportunity to allow or reject events before lower-priority modules.

10 Claims, 5 Drawing Sheets



OTHER PUBLICATIONS

InSoft Brochure: *Collaborative Features that Make the Difference,* InSoft®, (2 pages), (undated).

C. Petzold, *Programming Windows 3.1,* Redmond: Microsoft Press, 1992, pp. 10–15, 34–39.

Understanding TAO, Computer Telephony, vol. 4, Iss. 2, Feb. 1996, pp. 141–162.

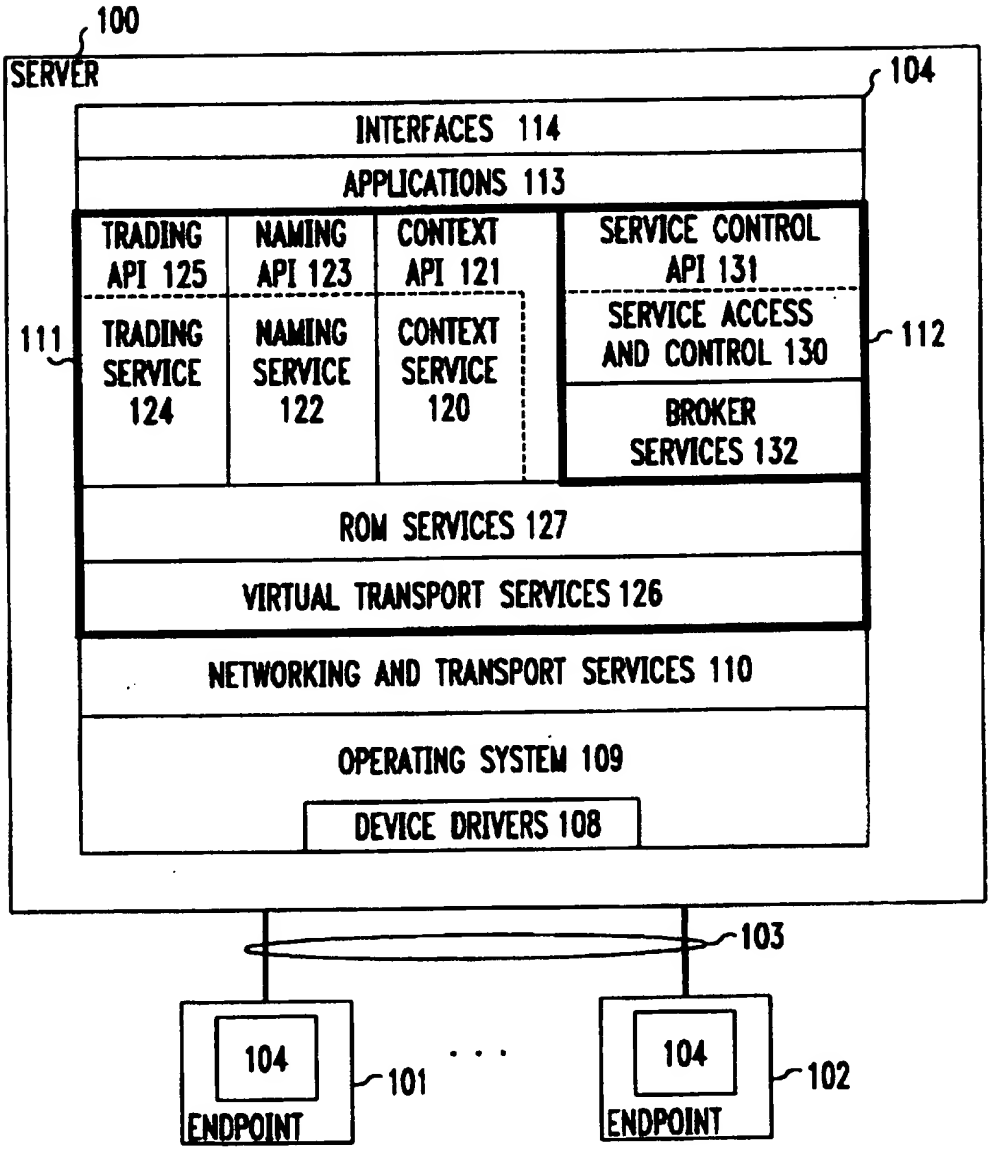
E. Margulies, *More Computer Telephony Architecture, Anyone?,* Computer Telephony, vol. 4, Iss. 3, Mar. 1996, pp. 228–230.

R. Grigonis, *The Origin of LAN Call-Control APIS,* Computer Telephony, Apr. 1995, pp. 102–105.

R. Grigonis, *Novell Netware Telephony Services,* Computer Telephony, Apr. 1995, pp. 83–84.

H. Newton, *The Firestorm Behind Mitel's "Open" PBX in a PC,* Computer Telephony, vol. 4, Iss. 1, Jan. 1996, pp. 10–22.

FIG. 1


100
SERVER
104
INTERFACES 114
APPLICATIONS 113
TRADING API 125
NAMING API 123
CONTEXT API 121
SERVICE CONTROL API 131
111
TRADING SERVICE 124
NAMING SERVICE 122
CONTEXT SERVICE 120
SERVICE ACCESS AND CONTROL 130
112
BROKER SERVICES 132
ROM SERVICES 127
VIRTUAL TRANSPORT SERVICES 126
NETWORKING AND TRANSPORT SERVICES 110
OPERATING SYSTEM 109
DEVICE DRIVERS 108
103
104
ENDPOINT
101
104
ENDPOINT
102

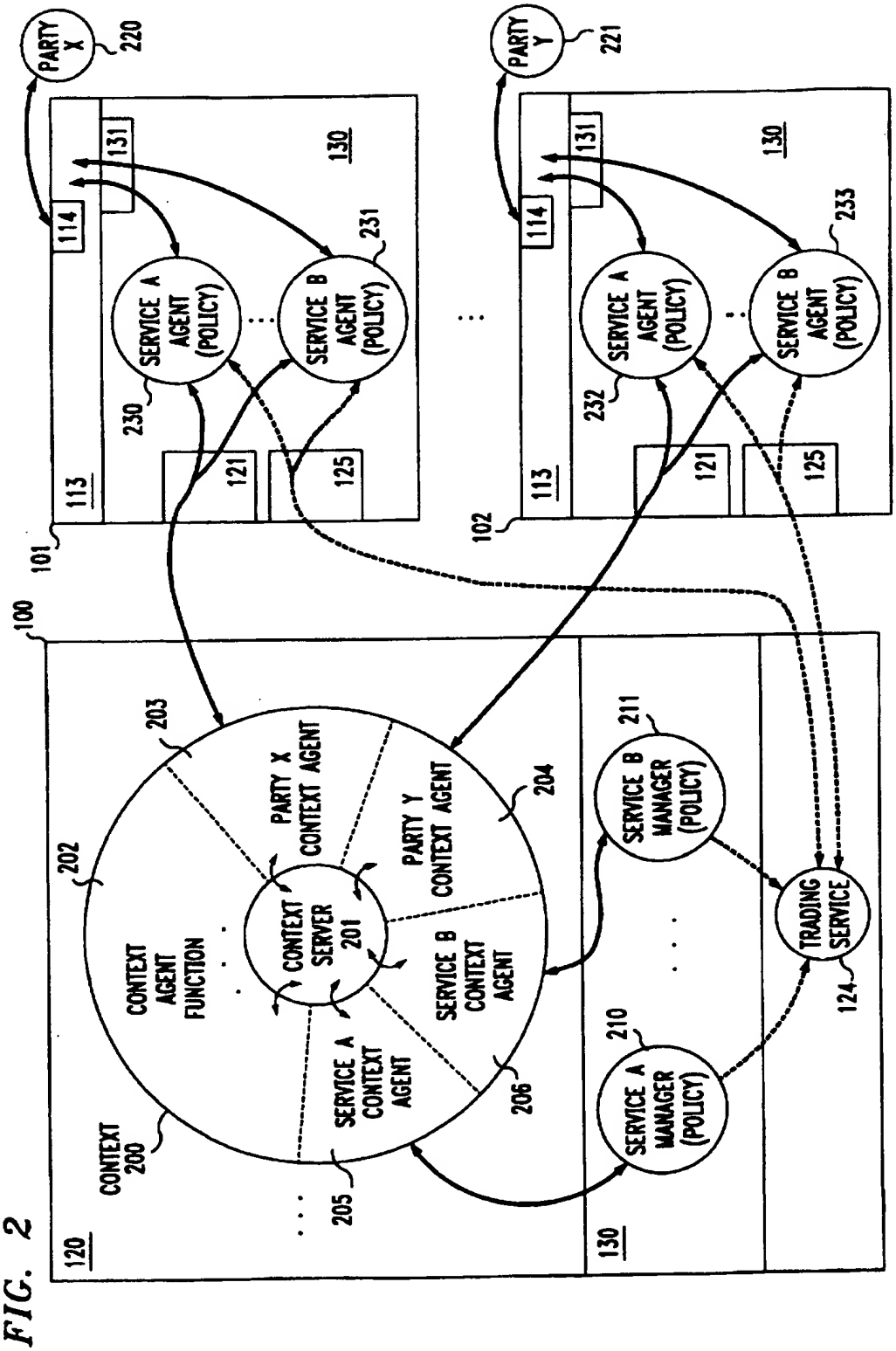

PARTY X
220
PARTY Y
221
114
131
130
113
121
125
SERVICE A AGENT (POLICY)
SERVICE B AGENT (POLICY)
230
231
232
233
101
102
100
120
CONTEXT 200
CONTEXT AGENT FUNCTION
CONTEXT SERVER 201
PARTY X CONTEXT AGENT
PARTY Y CONTEXT AGENT
SERVICE A CONTEXT AGENT
SERVICE B CONTEXT AGENT
202
203
204
205
206
SERVICE A MANAGER (POLICY)
SERVICE B MANAGER (POLICY)
210
211
TRADING SERVICE
124

FIG. 2

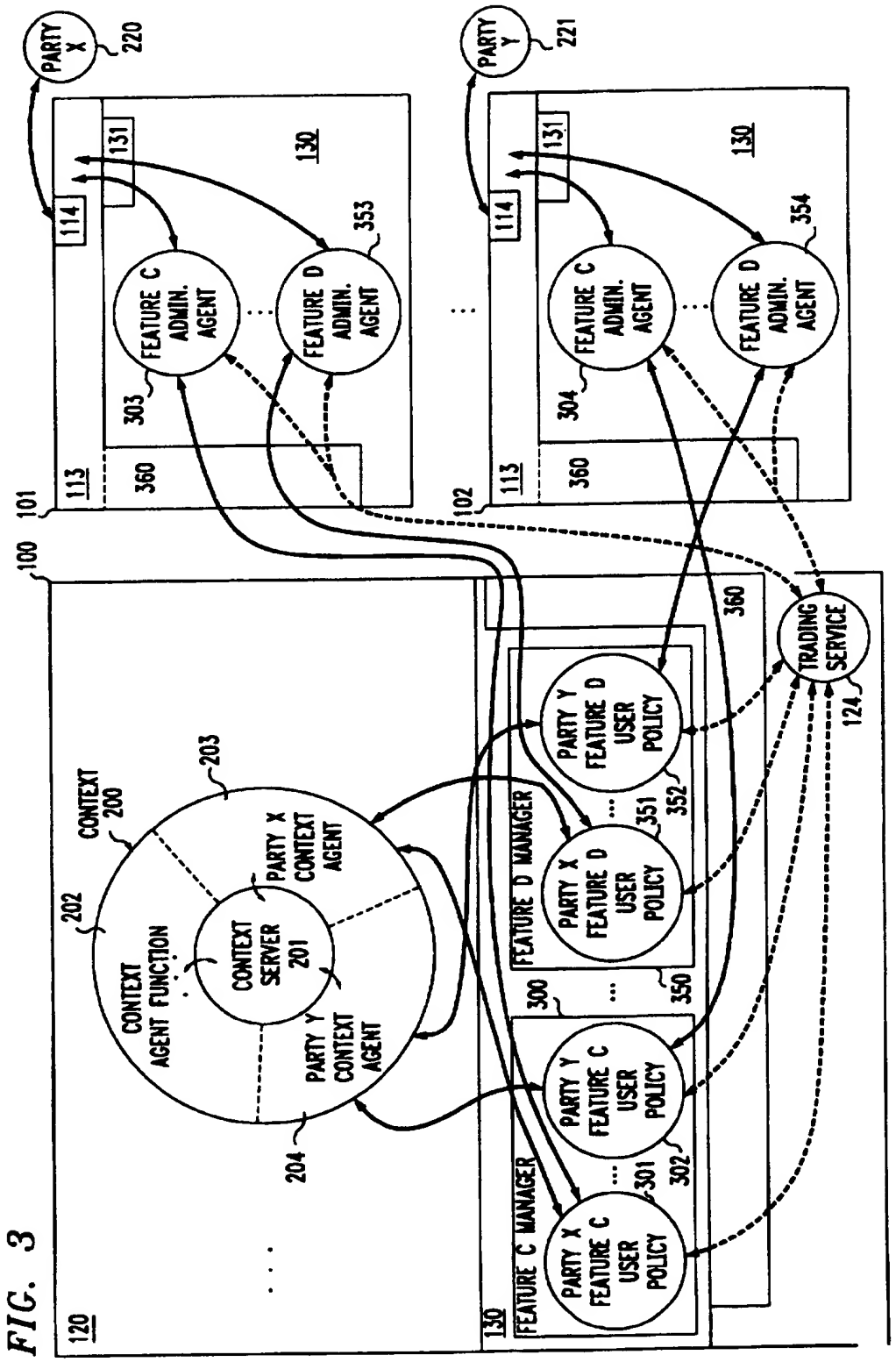

PARTY X
220
PARTY Y
221
114
131
130
FEATURE C ADMIN. AGENT
303
FEATURE D ADMIN. AGENT
353
304
354
113
360
101
102
100
120
CONTEXT 200
203
202
CONTEXT AGENT FUNCTION
CONTEXT SERVER 201
PARTY X CONTEXT AGENT
PARTY Y CONTEXT AGENT
204
FEATURE D MANAGER
FEATURE C MANAGER
PARTY Y FEATURE D USER POLICY
PARTY X FEATURE D USER POLICY
PARTY Y FEATURE C USER POLICY
PARTY X FEATURE C USER POLICY
352
351
350
300
302
301
TRADING SERVICE
124

FIG. 3

FIG. 4

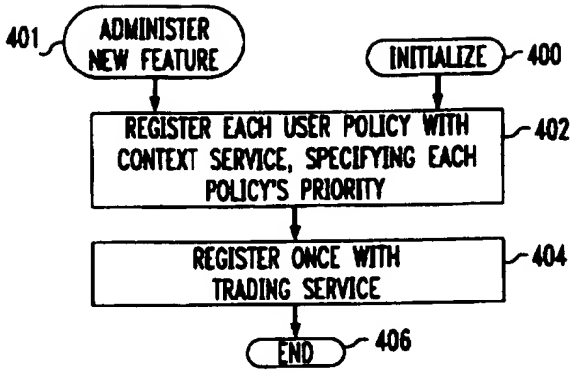


FIG. 5

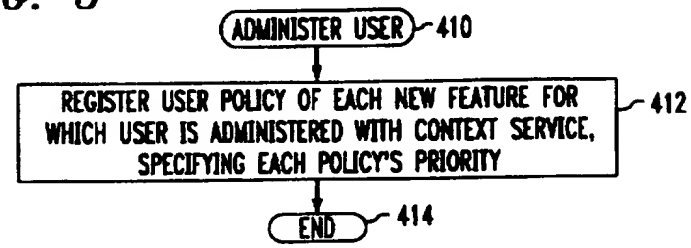


FIG. 6

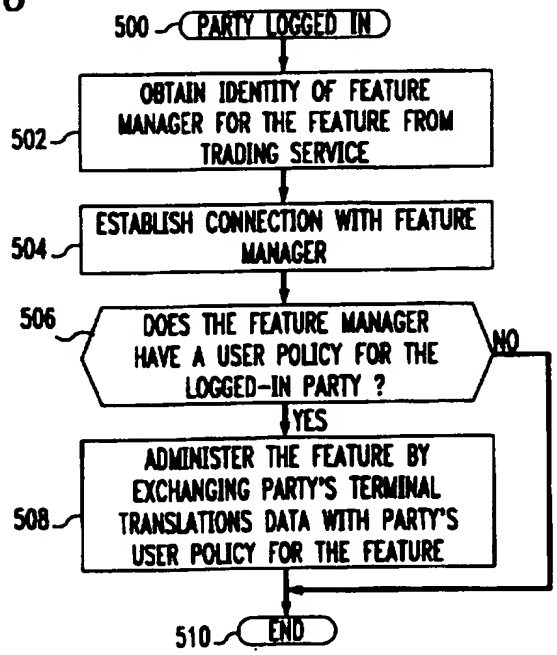


FIG. 7

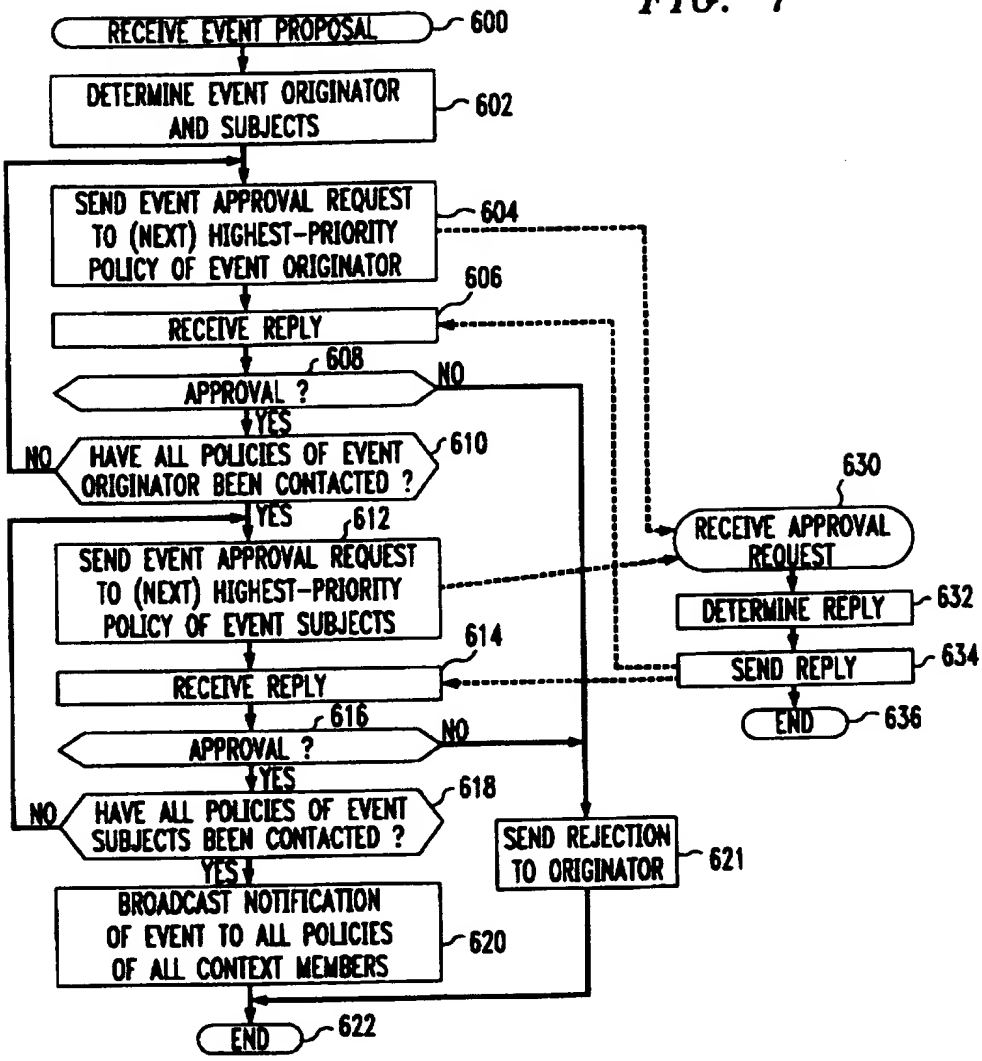
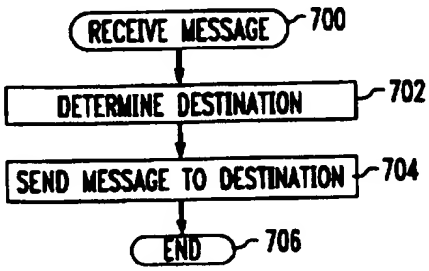

RECEIVE EVENT PROPOSAL
600
DETERMINE EVENT ORIGINATOR AND SUBJECTS
602
SEND EVENT APPROVAL REQUEST TO (NEXT) HIGHEST-PRIORITY POLICY OF EVENT ORIGINATOR
604
RECEIVE REPLY
606
APPROVAL ?
608
NO
YES
HAVE ALL POLICIES OF EVENT ORIGINATOR BEEN CONTACTED ?
610
NO
YES
SEND EVENT APPROVAL REQUEST TO (NEXT) HIGHEST-PRIORITY POLICY OF EVENT SUBJECTS
612
RECEIVE REPLY
614
APPROVAL ?
616
NO
YES
HAVE ALL POLICIES OF EVENT SUBJECTS BEEN CONTACTED ?
618
NO
YES
BROADCAST NOTIFICATION OF EVENT TO ALL POLICIES OF ALL CONTEXT MEMBERS
620
SEND REJECTION TO ORIGINATOR
621
END
622
RECEIVE APPROVAL REQUEST
630
DETERMINE REPLY
632
SEND REPLY
634
END
636

FIG. 8


RECEIVE MESSAGE
700
DETERMINE DESTINATION
702
SEND MESSAGE TO DESTINATION
704
END
706

ARRANGEMENT FOR FACILITATING PLUG-AND-PLAY CALL FEATURES

TECHNICAL FIELD

This invention relates to stored-program-controlled communications systems, including multimedia communications systems.

BACKGROUND OF THE INVENTION

Traditionally, adding calling features to switching-system (e.g., central office or PBX) control software has required modifying existing feature programs while paying careful attention to interactions between existing features and the new features. For example, implementing call-center features on a PBX that already provides call-forwarding and call-coverage requires careful coordination between the two sets of features, ensuring that calls are handled correctly when a call-center agent enables call-forwarding or call-coverage. In a multimedia telecommunicaions system where users are presented with graphical user interfaces, the need to provide new user interfaces to control new features poses an additional problem: in order for the endpoint user-interface and the server feature-software to communicate, the endpoint-to-server protocol generally needs to be updated.

The above-mentioned constraints present an imposing barrier to developers who are responsible for continuing maintenance and upgrades of the features. These constraints present an even-more imposing barrier to the development by third-party vendors of new features or of new versions of existing features. Third parties generally have neither the interest nor the capability to modify the telecommunications system software in order to make their feature software fully compatible therewith. Consequently, the telecommunications system owner is dependent exclusively on the system manufacturer for the feature set and feature upgrades of the system.

SUMMARY OF THE INVENTION

This invention is directed to solving these and other problems and disadvantages of the prior art. Generally according to the invention, there is provided a telecommunications system infrastructure that facilitates easy insertion of feature software into an existing said telecommunications system, and easy integration of the new calling features and their implementing software with existing features and their software.

Specifically according to one aspect of the invention, a call-control apparatus that executes feature-implementing software, wherein each call feature is implemented as a server program and a cooperating client program, has the following elements. An arrangement (illustratively a context service) for registering for a user an instance (illustratively a user policy) of a server program that implements any call feature, in a same manner as any other instances of any server programs that implement any call features, substantially at any time during operation of the call-control apparatus to provide the call feature for the user. In other words, the registration mechanism is identical for all feature-implementing programs, and operates dynamically. An administration interface (illustratively an application program interface, or API) for communicating information between the server program and a cooperating said client program used by the user, in a same manner as between any other client programs and any server programs that implement any call features, to customize the instance of the server program for the user. In other words, the administration interface is identical for all feature-implementing programs. An arrangement (illustratively a context, which is a cyberspace meeting room, along with the context service) for involving the instance of the server program in a call to which the user is a party, in a same manner as any other instance of any server programs that implement any call features, to provide the feature to the call. In other words, the mechanism for involving feature-implementing programs in calls is also identical for all features.

The above-characterized call-control apparatus provides all the interaction that is needed between feature-implementing programs as well as between feature-implementing programs and other service-implementing programs. Consequently, substantially any feature-implementing program that complies with the registration and conforms with the administration and in-call-involvement communications schemes may be added to and used in the apparatus; the call-control apparatus automatically effects integration of any such new feature-implementing program into the existing environment.

Preferably, interactions between feature-implementing programs are managed efficiently by having priorities associated with the individual programs. Each instance of each server program has a priority associated therewith, and the arrangement for involving feature-implementing programs in calls includes an arrangement (illustratively a context service and its context API) that responds to occurrence of a request for service (referred to herein as "an event") in the call by giving instances of server programs that are involved in the call each a chance to respond to the event in an order of their associated priorities. In other words, higher-priority programs are given an opportunity to approve or reject events before lower-priority programs. The system designer, administrator, or users are allowed to change the relative priorities of the feature-implementing programs and thereby control how features interact with each other. For example, it allows the administrator to specify whether call coverage is invoked for a call before call forwarding, or vice versa.

Preferably, the arrangement for involving feature-implementing programs in calls includes a second interface (illustratively also an API, referred to herein as the context API) for communicating events between a plurality of server programs that are involved in the call, in a same manner as between server programs that are involved in any other calls. In other words, the second communications interface is also identical for all feature-implementing programs. The arrangement responds to receipt through the second interface of a proposal of an event from a first server program—illustratively the originator of the proposed event—that is involved in a call by sending a request for approval of the event through the second interface to the first server program and also to other (at least one) second server programs that are also involved in the call. The arrangement then responds to receipt through the second interface of approval of the event from both the first and the second server programs by sending notice of the approval of the event through the second interface to both the first and the second server programs—and preferably to all server programs that are involved in the call—to cause the event to be effected. The arrangement preferably further responds to receipt through the second interface means of rejection of the event from either the first or the second server program by forbearing from sending an approval of the event to both the first and the second server programs to prevent the event from being effected, and preferably also sends the rejection through the second interface to the first server program to abort the event.

Specifically according to another aspect of the invention, a method of controlling calls in an apparatus wherein each call feature is implemented as a server program and a cooperating client program and which executes the client and the server programs, comprises the following steps. In response to a user becoming entitled to a call feature, an instance of the server program that implements the call feature is registered for the user, in a same manner as any other instance of any server programs that implement any call features, substantially at any time during operation of the apparatus, to provide the call feature for the user. Then in response to the user using the client program, information is communicated between the server program and the client program through an administration interface, in a same manner as information is communicated through the administration interface between any other client programs and any server programs that implement any call feature, to customize the instance of the server program for the user. Then in response to the user becoming a party to a call, the instance of the server program is involved in the call, in a same manner as any other instance of any server programs that implement any call features, to provide the feature to the call. Plug-and-play feature capability is thereby effected for feature-implementing software.

These and other advantages and features of the present invention will become more apparent from the following description of an illustrative embodiment of the invention taken together with the drawing.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of an illustrative multimedia communications system;

FIG. 2 is a block diagram of a call-service model implemented by the system of FIG. 1;

FIG. 3 is a block diagram of a call-feature model implemented by the system of FIG. 1, which embodies an illustrative implementation of the invention;

FIGS. 4 and 5 are functional flow diagrams of registration procedures of feature managers and their user policies of the model of FIG. 3;

FIG. 6 is a functional flow diagram of a feature administration procedure of a feature administration agent of the model of FIG. 3;

FIG. 7 is a functional flow diagram of an event-negotiation procedure of a context and context members of the model of FIG. 3; and

FIG. 8 is a functional flow diagram of a message-passing procedure of the context of the model of FIG. 3.

DETAILED DESCRIPTION

FIG. 1 shows one possible architecture of a multimedia communications system. The system comprises a plurality of communications endpoints **101–102** connected by communications links **103** with a communications server **100**. The system of FIG. 1 can be, for example, a telephone system where server **100** comprises a multimedia-enabled switching system such as a Lucent Technologies Definity® G3 PBX, endpoints **101–102** comprise video workstations such as the NCR Vistium® stations, and links **103** comprise high-bandwidth telephone lines such as ISDN BRI lines. However, in this illustrative example, server **100** is assumed to be a multimedia-services server such as the Lucent Technologies Inc. MMCX multimedia communications exchange, endpoints **101–102** are assumed to be multimedia (including video) workstations, and links **103** are assumed to be a local-area network (LAN) or a wide-area network (WAN). Server **100** and endpoints **101–102** are stored-program-controlled entities. As such, each includes a memory for storing control software, a processor for executing the stored control programs, and input and output interfaces to the outside world, as is well known and understood in the art.

According to well-known software-system design principles, the control software **104** of server **100** and endpoints **101–102** is organized in a multi-layer hierarchy. At the lowest level in the software hierarchy, the control software of server **100** and endpoints **101–102** in this illustrative example comprises a conventional operating system **109**—such as the Lynx® operating system—that includes conventional device drivers **108**. Next in the hierarchy is a conventional networking and transport services layer **110**—such as the Transmission Control Protocol/Internet Protocol (TCP/IP)—which provides the information-movement (i.e., control-signal and information-signal transmission) services between server **100** and endpoints **101–102**. Built on top of layer **110** is a middleware layer **111**. Middleware is a term for a software platform that provides network-transparent support for the development and implementation of network-based distributed-system applications (e.g., communications services). It is both an applications-development tool and a run-time environment. It provides a distributed object-based computing infrastructure including distributed object life-cycle management, network abstraction, and operating-system and transport-service virtualization. It therefore allows communications applications to be written independently of the resident operating system, the network transport, the interworking algorithms, etc. It also supports a middleware services layer **112** which provides common services that support various communications applications, such as services for session management, routing, event collection, service location, etc. Implemented on top of layers **111** and **112** are applications **113**, e.g., specific communications services programs. Applications **113** communicate with layers **111** and **112** by means of application program interfaces (APIs) of layers **111** and **112**, and communicate with users and/or administrators via interfaces defined by an interfaces layer **114**. In the case of endpoints **101–102**, applications **113** illustratively comprise a version of Insoft's Communique!™ collaboration software.

Layers **111** and **112** illustratively comprise the communications middleware software of the Lucent Technologies Inc. MMCX, heretofore known as CoMMware. Layer **111** comprises the middleware platform, while layer **112** comprises middleware-compliant service components that make use of the middleware platform primitives to control calls and their different-media components and to supply calling features (like call-coverage and call-forwarding, for example.) The service components include service managers (servers) and service agents (clients).

The middleware platform provides an infrastructure for bringing parties and multimedia services into communications "contexts" which provide bases for negotiation of service parameters. Each communications session (e.g., a multimedia call) is represented by its own context. The architecture provides support for customizable service negotiation and control software, called "policies", that allows application and service developers to meet a wide variety of product-and service-specific needs.

In the model of communications that is presented by the middleware, all communications take place within a context, and parties and services are associated with one another as

members within the context by a context service. The context service is somewhat analogous to Microsoft Corporation's Windows™ system. Just as the Windows system distributes events that reflect a change in the applications' presentation environment to all applications running in that environment, so does the context service distribute events which reflect a change in the communications context to all members of that context. In addition to the event-notification mechanism, the context service also supports message-passing among context members, for example, to enable negotiation of interworking parameters between endpoints and servers with possibly-disparate capabilities.

The middleware effectively provides a signaling overlay on top of the underlying network architecture, which overlay supports multiparty, end-to-end negotiation that facilitates the design of interoperable multimedia communications products and services. Middleware concepts of context, virtual transport, and trading aid in the provisioning of multiparty, multimedia distributed communications in heterogeneous environments.

The model of communications that is presented by the middleware is shown in FIG. 2. The middleware facilitates bringing parties and services together in a "cyberplace", which is referred to as context **200**. A context server **201** manages context **200** to/from which may be added/dropped the context members. Members may be parties and services. A logged-in user of an endpoint **101–102** who is a member of a context is referred to as a party (**220–221**) to that context. A service is represented in a context **200** by its service manager **210–211**. Parties and services are treated identically by context server **201**, and are referred to simply as "members". All members in a context **200** are represented by a context agent facility **202**. Each member of context **200** is logically represented in context agent facility **202** by its own corresponding member context agent **203–206** (e.g., its own virtual port on context agent facility **202**). When context **200** changes as a result of members being added to or dropped from the context, context server **201** alerts all members' context agents **203–206**, which in turn notify their corresponding members. When a new member joins an existing context **200**, all members already in the context are similarly notified, and each has a chance to exchange some initial "get acquainted" messages with the new member and with other members that were already in the context. In middleware, this is called "negotiation", since it is generally used to achieve a common ground for communications between the members (parties and services) in the context.

The middleware provides support for brokering in three ways. First, the middleware includes trading service **124**, which is a database system that can be used to locate services based on service characteristics. Services are constructed in a client/server configuration, with programs that actually provide the services, called service managers **210–211** (also call services, service components, resource or media servers, or resource or media managers), being located in MMCX server **100**, and programs that obtain the services from service managers **210–211** on behalf of applications in endpoints **101–102**, called service agents **230–233** (also called service clients, or resource or media agents), being located in endpoints **101–102**. Service managers **210–211** can register with trading service **124**, giving their service attributes and capabilities. Trading service **124** provides a query capability to enable service agents **230–233** to obtain identities of services (i.e., of service managers **210–211**) that can meet the common needs of the parties in a context. Secondly, specialized brokering services **132** can be written, which are servers themselves that can be brought into a context. A brokering service uses the generic negotiation mechanism provided by the middleware to gather the service-related attributes of the parties in the context, enabling it to bring other service managers **210–211** into the context that can meet their, perhaps diverse, needs. And finally, the middleware supports the development of implicit brokers which, as policies of context server **201**, can examine the attributes of the context and its members to bring services into the context. This sort of broker might be used, for example, to bring billing services into the context. These brokering mechanisms can also be used in unison. A specialized broker may, for example, gather parties' attributes and formulate a complex query to trading service **124** to locate the right service.

The middleware provides a framework for introducing a level of signalling and control for communications sessions that fits logically above the transport network. This means that software can be written to formalize communications that are required to set up calls. The middleware supports codification of the signalling used for service composition and separates it from that used for control of bearer channels and network connections. A member context agent **203–206** of context agent facility **202** utilizes virtual transport to access underlying transport services for establishing a signalling connection to a context server **201**, which, in turn, is then able to establish signalling connections with other member context agents **203–206**. Context agent facility **202** utilizes a transaction protocol with context server **201** to create a context **200** for a communications session and to associate parties **220–221** and service managers **210–211** with the context. Integral to this transaction protocol, the middleware provides a foundation for negotiation among parties' service agents **230–233** and service managers **210–211** which allows media-specific service agents **230–233** and service managers **210–211** to agree on service-specific parameters regarding the communications session. The specific negotiation protocol, as defined in common for a specific media service, is implemented in replaceable program entities (policies) which are bound to context transaction processing.

While the communications model supports familiar communication system features (with parties and transport), more elaborate communications in which multiple parties and a rich array of services are added and removed dynamically are also supported naturally within the model. For example, a two-party voice call can be turned into a multiparty conference with a video and a multipoint shared application by adding additional parties, a video-connection service, and a shared-data service to the context. Further, since the services in a context may be independent of one another, each can be added and removed at any time without affecting the others. These attributes of the model result from the concept of context and the fact that the signalling for bearer-channel connection-control and for establishment and control of the context are separate.

A second example illustrates additional attributes of this model. If an interactive service (such as "800"-number video-catalog shopping) were desired, an endpoint would be able to request the service and negotiate the attributes of the service to conform to its own capabilities. But then, the service itself could request that required ancillary services be added to the context, such as billing, order processing, and credit card authorization. This illustrates the fundamental symmetry of the middleware architecture that provides parties and services with the same status in a context, thus allowing all members the full power of the context transaction protocol.

As shown in FIG. 1, the middleware is constructed as follows. There are six architectural elements to the middleware platform:

Context service 120: The context service provides the supporting mechanisms for the middleware model of communications that provides a context for a communications session in which service providers and service users are treated as undifferentiated members with equal privileges and capabilities. Context service 120 is provided by interactions between a context server 201 that manages a context 200 and a context agent 202 that represents the members of the context. Communications with context service 120 are effected through a context API 121. Context API 121 is available to both applications software and service modules. This means that, although policy modules will normally buffer applications from context transactions, negotiation, and service control, it is possible for applications to directly react to and influence these activities.

Naming service 122: A distributed naming database that allows the middleware and middleware-based applications to access transport addresses associated with a middleware identifier (CWID). Each service manager 210–211 and party 220–221 has its own CWID. The naming service performs two mappings: 1) a mapping from a CWID to a transport-independent address (a virtual transport address, or VTA), and (2) a mapping from a VTA to transport-dependent addresses and attributes. The attributes associated with each VTA illustratively consist of "attribute name; attribute value" pairs, where there is a fixed set of attribute names supported. Service agents 230–233 use the first mapping to get the VTA for a given party or service manager and then give the resulting VTA to virtual transport service 126, which calls on naming service 122 to perform the second mapping in order to obtain actual transport addresses for establishing transport connections to these parties and service managers. Communications with naming service 122 are effected through a naming API 123.

Trading service 124: Trading in the middleware is service selection based on combined attributes of the members of a context. Trading service 124 is a database that supports service registration and the ability to locate service managers 210–211 by required attributes. Trading service 124 has the ability to satisfy queries from service agents 230–233 that require it to find the "best match" of party attributes to service attributes. Service managers register with the trading database. Brokers can be developed in the middleware that use trading service 124 to find a best match for the collective needs of the members of a context. Communications with trading service 124 are effected through a trading API 125.

Remote object management (ROM) service 127: The ROM service is a simple object request broker (in the object-oriented programming sense). It uses virtual transport service 126 to allow object methods to be invoked remotely. ROM service 127 is available to both the middleware itself and to applications and policy modules. Policy modules may make use of ROM service 127 to establish out-of-context communications channels with peer or server policy modules. Applications make use of ROM service 127 to establish client-server connections.

Virtual transport service 126: An abstraction of transport that presents a common model for a variety of communications networks. The use of virtual transport enhances the portability of applications and services and their interoperability in heterogeneous network environments. Each entity in the middleware is given a virtual transport address (VTA) which allows addressing of and connecting to that entity in a network-independent manner.

In addition to these elemental services, the middleware provides a programming framework and associated libraries to facilitate development of run-time libraries that implement protocols for middleware-compliant service access and control 130 and brokering services 132. The program entities that are developed within this framework are objects (in the object-oriented programming sense), called policy modules or policies, that implement service and access control 130 and brokering services 132, and constitute the client/server software that provides services and service access. In other words, service managers 210–211 and service agents 230–233 are policies that perform the service-specific negotiation and control functions that are required for service delivery.

To control independently a dynamic mixture of services, the concept of context provides a place to instantiate a locus of control for the composition of these services and facilitates the multi-way negotiation needed to deliver the services to a variety of endpoints. This requires that the detailed attributes of various media services (feature control mechanisms, encoding choices, transport requirements, delay and synchronization characteristics, etc.) be understood and agreed-to by service providers (service managers 110–112) and service users (parties 220–221).

The middleware introduces the idea of "negotiation" among members, typically between parties and service managers, to allow services to be provided in a manner that ensures compatibility and consistency of service delivery to the parties. The middleware provides a framework for incorporating policy modules into a system that are available for use by applications for performing service-specific negotiation in reaction to changes to the context. Policy modules can also be used during service delivery to provide service-control functions, e.g., to tell a video server which video stream to send. Policy modules are essentially service-specific run-time libraries that implement service-specific negotiation and control protocols. Communications by applications 113 with policy modules of service access and control 130 are effected through a service control API 131.

Context service 120 with appropriate policy modules enables deployment of new multimedia services without having to enhance underlying network equipment. Naming service 122 and trading service 124 also facilitate service composition by enabling applications to locate the services that are needed to meet the needs of the members in a context. Brokering services 132 can be created that perform the function of gathering up appropriate party attributes, formulating the required trader query, and inviting the returned service manager into the context. These brokers generally are service-type specific (e.g., audio, video, shared application), and are implemented as separate services that can be added to a context. Broker agents work with the brokering server to locate the needed service manager and add it to the context. In some cases, a broker may be implemented as a policy module of context server 201 which, by virtue of its ability to eavesdrop on all context transactions, can perform a service-location function. Naming service 122 complements the brokering service by providing a facility for converting a middleware entity name to the transport network attributes required to connect to that entity.

Naming service 122 serves both applications 113 and virtual transport service 126. Applications typically use naming service 122 to map a qualified CWID (e.g., an

E.164-conformant address, such as a phone number) into a virtual transport address (VTA, e.g., also an E.164-conformant address). The VTA appended with a logical-port identifier is "handed" to virtual transport service **126** which uses naming service **122**, once again, to map the qualified VTA to transport-specific attributes. For example, say that a user has a CWID of 303.538.4071, then naming service **122** would be used by a context server **201** to map 303.538.4071.context_port to the VTA of that user's member context agent, which might be 303.538.4000. When the context server **201** wishes to establish a connection to the context agent's port for accepting context messages, it asks virtual transport service **126** to establish a connection to 303.538.4000.context_port. Virtual transport service **126** would, in turn, use naming service **122** to map 303.538.4000.context_port to the transport specific address (es) of the appropriate virtual port on context agent **202**.

This description of the middleware applies fundamentally to both MMCX server **100** and endpoints **101–102**, although APIs **121**, **123**, and **125** are the only portions of services **120**, **122**, and **124** that are used on endpoints **101–102**. A distributed client/server architecture is utilized whereby client software (service agents **230–233**) in an endpoint **101–102** works cooperatively with server software (service managers **210–211**) in server **100** to provide the brokering services as well as the media-control services which provide the value-added communications services to users.

According to the invention, the above-described infrastructure and call model are used to provide support for plug-and-play call features in the manner illustrated in FIG. **3**.

Feature software is implemented using the client/server architecture. As shown in FIG. **3**, each feature comprises a feature manager (a server module) **300**, **350** on MMCX server **100** and a plurality of feature administration (admin.) agents (client modules) **303–304**, **353–354**, one on each endpoint **101–102**. Each feature manager **300**, **350** is implemented as one or more user policies **301–302**, **351–352**, one for each user **220–221** that is entitled to use the feature. Each user policy **301–302**, **351–352** is one user's customized instance of the feature manager **300**, **350**, respectively. Feature managers **300**, **350** and their user policies that are involved in a call are not members of that call's context **200**, but each communicates with context **200** through the party context agent **203**, **204** of its corresponding party **201**, **202**. User policies of the same and of different feature managers **300**, **350** keep track of each other through the event-notification mechanisms provided by context service **120**, but they generally do not communicate with each other directly through an out-of-context exchange of messages.

Unlike the feature managers, feature admin. agents **303–304**, **353–354** are not implemented as policies. Feature admin. agents do not execute features, but rather only turn features on and off at endpoints **101–102** and communicate administrative information between parties **220–221** and feature managers **300**, **350**. Feature admin. agents and their corresponding feature managers communicate with each other outside of context **200** through an administration API **360** that is implemented at the applications layer **113**. Administration API **360** uses trading API **125** and ROM service **127** to locate, and to communicate with, feature managers **300**, **350**. Graphical user interfaces **114** and feature admin. agents for any number of features can be added to (or deleted from) any endpoint **101–102** at any time, and administration API **360** enables the interfaces and agents to locate, and to communicate with, the corresponding feature managers.

In order to make context service **120** aware of their existence, the user policies of feature managers **300**, **350** must register with context service **120**. The registration procedure is shown in FIGS. **4** and **5**. Upon its own creation by an administrator of MMCX server **100**, at step **401** of FIG. **4**, or upon initialization of MMCX server **100** (whichever is earlier), at step **400**, a feature manager **300**, **350** registers separately each of its user policies **301–302**, **351–352** with context service **120** of MMCX server **100**, at step **402**. As part of each registration with context service **120**, a feature manager **300**, **350** specifies the priority that has been administered for each user policy. The priorities for all user policies of a feature manager may be all the same, or they may be different. When the priority of one or more of its user policies is administratively changed, the feature manager re-registers with context service **120** with the new priorities. A feature manager **300**, **350** also registers once with trading service **124** of MMCX server **100**, at step **404**, before concluding registration, at step **406**.

Subsequently, at any time whenever a new user is administered for features or an existing user is administered for a new feature, at step **410** of FIG. **5**, each feature manager **300**, **350** of the user's new features registers the user's new user policies **301–302**, **351–352** with context service **120** of MMCX server, at step **412**. As part of the user policies' registration with context service **120**, each feature manager **300**, **350** specifies the priority that has been administered for the new user policy. Registration of the new user policies then concludes, at step **414**.

FIG. **6** shows the feature-administration procedure for feature admin. agents **303–304**, **353–354**. When a party **220**, **221** logs into an endpoint **101**, **102**, at step **500**, each feature's feature admin. agent of the logged-into endpoint uses trading service **124** to find the identity of the corresponding feature manager **300**, **350** for the corresponding feature, at step **502**. The feature admin. agent then uses the identity to establish a communications connection with the corresponding feature manager via administration API **360**, at step **504**, and determines from the feature manager whether there is a corresponding user policy for the logged-in party **220**, **221**, at step **506**. If there is not a user's policy for the logged-in party, the logged-in party is not authorized to use the corresponding feature, and so the feature admin. agent ends feature administration, at step **510**. If a corresponding user policy is found and its identifier is returned by feature manager **300**, **350** to the feature admin. agent, the logged-in party is authorized to use the corresponding feature, and so the feature admin. agent and the party's user policy then administer the feature for the party by exchanging that party's "terminal translations" data (as that term is commonly used in the telephony art), at step **508**, before ending administration, at step **510**. The terminal translations data that is provided by the party's user policy will have been pre-administered by an administrator of server **100**, while the terminal translations data that is provided by the feature admin. agent is obtained by feature admin. agent through interaction with the logged-in party through the feature's corresponding one of the interfaces **114** on the endpoint.

Consider, for example, the administration for the call-forwarding feature. The call-forwarding user interface on the endpoint invokes the feature administration API **360**, passing it the call-forwarding feature identifier, the identifier of the user who is administering call-forwarding, and a collection of call-forwarding administration data, including the identifier of the forwarded-to party and whether call-forwarding should become enabled or disabled. The feature

administration API **360** queries trading service **124** for the identifier of the feature server that matches the call-forwarding feature identifier, i.e., the call-forwarding feature server. If trading service **124** successfully returns this identifier, the administration API **360** sends the call-forwarding feature server the user's identifier and the call-forwarding administration data. The call-forwarding feature server receives the data and stores it in its feature translation database. Henceforth, during call processing, the administration data can be retrieved from the feature translation database and used to effect the call-forwarding feature.

For purposes of event-negotiation, event context **200** provides for three types of inter-member communications: proposals from initiators of events to context, event requests for approval from context to interested policies (those affected by the event), and event notifications from context to all context members. Any proposed event that causes a change in context or a change of state of a context member must be approved by the policies of the acting context member and the policies of the members being acted on. For example, a request to create a context is an event that must be approved by all policies of the requestor; a request to add or delete a member to or from a context or to change a member's state is an event that must be approved by all policies of the requestor and the subject of the request; and a request to destroy a context is an event that must be approved by all policies of the requestor.

FIG. **7** shows the event-negotiation procedure. When an event proposal is received by context **200** from a policy, at step **600**, context **200** determines from the event which context members need to approve the event, at step **602**, and then sends event requests for approval to the policies of each member which needs to approve the event—first to the requestor's policies serially and sequentially in the order of the policies' priorities, and then to the policies of the subject or subjects of the request serially and sequentially in the order of the policies' priorities. Context **200** first sends an event-approval request to the highest-priority policy of the event-originator, at step **604**. Upon receipt of an event request for approval, at step **630**, each policy that is asked for approval of the event executes whatever algorithm it has been programmed with to determine its approval or rejection, at step **632**, and sends its reply to context **200**, at step **634**, before concluding, at step **636**. Context receives the policy's reply, at step **606**, and checks whether it is an approval or a disapproval, at step **608**. If any policy of any member who needs to approve the event replies with a rejection of the event, as determined at step **608**, context **200** notifies the policy that made the event proposal, at step **621**, then ends, at step **622**, and the event is not effected. If the received reply is an approval, context checks whether it has sent an event-approval request to each policy of the event originator, at step **610**. If not, context **200** returns to step **604** to send the event-approval request to the next-highest priority policy of the event originator. If all policies of the event originator have been sent event-approval requests, context **200** proceeds to steps **612–618** to repeat the procedure of steps **604–618** for each policy of the event subjects. If all policies of all context members who need to approve the event reply with an acceptance of the event, as determined at step **618**, context **200** broadcasts notification of the event to all policies of all members of the context, at step **620**, so that the policies can implement, or take note of, the event, and ends, at step **622**. The event notifications at step **620** are not prioritized.

The policy priorities do not affect information messages, which are data messages or requests for information sent between clients and servers through APIs such as administration API **360**, for example. Their handling by APIs is shown in FIG. **8**. Upon receiving an information message, at step **700**, an API determines the message's destination, at step **702**, and forwards the message to that destination, at step **704**, before ending, at step **706**.

To provide the above-described communications capabilities, the various APIs include the following messages.

To allow an administrator of MMCX system **100** and parties **220–221** at endpoints **101–102** to administer the features, administration API **360** provides a message that comprises one function (command) and three arguments. The function is "administration request". The arguments are the identifier of the feature that is to be administered, the identifier of the party for which the administration is being done, and the administration data for use by the message recipient.

To enable feature managers and service managers to register with context service **120**, context API **121** provides a message that comprises one function and four arguments. The function is "policy registration request". The arguments are the identifier of the feature or service, the user policy that is registering (the registrant), the name of the party represented by the registrant, and the registrant's priority.

To enable feature managers and service managers to register with trading service **124**, trading API **125** provides a message that comprises one function and two arguments. The function is "manager registration request". The arguments are the identifier of the feature or service that is registering, and the CWID of the registrant.

To enable feature admin. agents and server agents to find suitable user policies and service managers via trading service **124**, trading API **125** provides two messages. One message has a function of "server request" from the agent to trading service **124**, and arguments of feature or service ID and feature or service parameters being requested by the agent. The other message has a function of "server response" from trading service **124** to the agent, and an argument of CWID of the feature's user policy or the service manager selected by trading service **124**.

To enable policies to initiate events, context API **121** provides a message that comprises any one of six functions and four arguments. The functions are "create context", "destroy context", "add member", "drop member", "change state", and "send message". The arguments are the CWID of the initiator, the new state in the case of the "change state" function, the CWID of the event subject for the functions other than "create context" and "destroy context", and the reason for the event.

To enable event requests for approval to be made, context API **121** provides a message that comprises one function and four arguments. The function is "request for approval". The arguments are the function and arguments of the event-initiating message.

To enable event approvals and denials, and notification of denials to event originators, context API **121** provides a message that has one function and one argument. The function is "approval reply". The argument is either approval or a failure code.

To enable event notifications, context API **121** provides a message that comprises one function and four arguments. The function is "event notification". The arguments are the function and arguments of the event-initiating message.

To enable in-context communication among context members, context API **121** provides a message that has one function and four arguments. The function is "send mes-

sage". The arguments are the CWID of the destination party, the ID of the feature or service being communicated with, the ID of the user policy being communicated with, and the message data. If the policy ID is omitted, the message is sent to all user policies of the identified feature or service.

To allow information-passing between user policies and feature admin. agents, administration API **360** provides a message that comprises one function and one argument. The function is "administer". The argument is an arbitrary data field.

Given this infrastructure, almost anyone can create a feature (that is, feature software) for use therein. Substantially the only requirements placed on the feature software are that (a) it have a client/server construction, that is, be constructed as a feature-manager server module plus a feature admin. agent client module and (b) be compliant with messages of the interfaces (context API **121**, administration API **360**, and trading API **125** in this example) that were enumerated above, and with their functions. Compliance with requirements (a) and (b) above ensures that the feature can both operate in the system of FIG. **1** and inter-operate with other features and services of the system of FIG. **1**. In other respects, the internal structure and functionality of the feature are irrelevant to the system of FIG. **1** and its various components, both hardware and software.

Of course, various changes and modifications to the illustrative embodiment described above will be apparent to those skilled in the art. For example the context service need not be provided by a single server, but may be distributed across a network of servers. Also, the industry-standard Object Management Group's (OMG's) object request broker (ORB) could be used instead of the remote object management (ROM) services and the virtual transport (VT) services. Furthermore, the middleware and its functions could be distributed over a plurality of servers. Also, additional functions and messages may be added to the APIs. Or in addition to, or instead of, the context API protocol, the system can support other multimedia communication protocols (e.g., enhanced H.323, H.320, T.120, conventional audio telephony protocols). Such changes and modifications can be made without departing from the spirit and the scope of the invention and without diminishing its attendant advantages. It is therefore intended that such changes and modifications be covered by the following claims.

The invention claimed is:

1. A call-control apparatus wherein each call feature is implemented as a server program and a cooperating client program, comprising:

means for executing the client and the server programs;

means for registering for a user an instance of a server program that implements any call feature, in a same manner as any other instances of any server programs that implement any call features, substantially at any time during operation of the call-control apparatus to provide the call feature for the user;

an administration interface for communicating information, between the server program and a cooperating said client program used by the user, in a same manner as between any other client programs and any server programs that implement any call features, to customize the instance of the server program for the user; and

means for involving the instance of the server program in a call to which the user is a party, in a same manner as any other instance of any server programs that implement any call features, to provide the feature to the call.

2. The apparatus of claim **1** wherein:

each instance of each server program has a priority associated therewith; and

the means for involving comprise

means responsive to occurrence of an event in the call, for giving instances of server programs that are involved in the call each a chance to respond to the event in an order of their associated priorities, to control interaction between the features implemented by the instances of the server programs that are involved in the call.

3. The apparatus of claim **1** wherein

the means for involving comprise:

a second interface for communicating events between a plurality of server programs that are involved in the call, in a same manner as between server programs that are involved in any other calls;

means cooperative with the second interface, responsive to receipt through the second interface of a proposal of an event from a first server program that is involved in the call, for sending a request for approval of the event through the second interface to the first server program and to a second server program that is also involved in the call;

means cooperative with the second interface and responsive to receipt through the second interface of approval of the event from both the first and the second server program, for sending the approval of the event through the second interface both to the first server program and to the second server program to cause the event to be effected, and responsive to receipt through the second interface of rejection of the event from either the first or the second server program, for forbearing from sending the approval of the event both to the first server program and to the second server program to prevent the event from being effected.

4. The apparatus of claim **3** wherein:

each server program has a priority associated therewith, and

the means for sending a request for approval are responsive to the receipt of the event, for sending the request for approval to the first and the second server programs serially in an order of the priorities of the first and the second server programs, to control interaction between the features implemented by the first and the second server programs.

5. The apparatus of claim **4** further comprising:

means for administratively changing the priorities that are associated with server programs, to change the interaction between the features implemented by the server programs.

6. The apparatus of claim **1** wherein:

the administration interface communicates information between the server program and the cooperating client program used by the user to customize a registered said instance of the server program for the user; and

the involving means involve a customized registered said instance of the server program in the call to which the user is a party.

7. The apparatus of claim **1** wherein:

each feature's server program comprises a plurality of instances of the server program, one for each user who is entitled to the feature.

8. The apparatus of claim **1** wherein:

the executing means comprise

means for executing the server programs, and

at least one means for executing the client programs;

the server programs, the means for executing the server programs, the registering means, and the involving means are included in a call controller;

an instance of the client program of each of at least some of the features and one of said means for executing the client programs are included in each of at least one call endpoint; and

the administration interface interfaces the at least one call endpoint with the call controller to communicate information between server programs on the call controller and the instances of the client programs on the at least one endpoint.

9. A method of controlling calls in an apparatus wherein each call feature is implemented as a server program and a cooperating client program and which executes the client and the server programs, comprising the steps of:

in response to a user becoming entitled to a call feature, registering for the user an instance of the server program that implements the call feature, in a same manner as any other instance of any server programs that implement any call features, substantially at any time during operation of the apparatus, to provide the call feature for the user;

in response to the user using the client program, communicating information between the server program and the client program through an administration interface, in a same manner as information is communicated through the administration interface between any other client programs and any server programs that implement any call feature, to customize the instance of the server program for the user; and

in response to the user becoming a party to a call, involving the instance of the server program in the call, in a same manner as any other instance of any server programs that implement any call features, to provide the feature to the call.

10. The method of claim 9 wherein:

each instance of each server program has a priority associated therewith; and

the step of involving comprises the step of

in response to occurrence of an event in the call, giving instances of server programs that are involved in the call each a chance to respond to the event in an order of their associated priorities, to control interaction between the features implemented by the instances of the server programs that are involved in the call.

* * * * *

US005323452A

United States Patent [19]

Dickman et al.

[11] Patent Number: 5,323,452

[45] Date of Patent: Jun. 21, 1994

[54] VISUAL PROGRAMMING OF TELEPHONE NETWORK CALL PROCESSING LOGIC

[75] Inventors: Bernard N. Dickman, Middletown; Nancy E. Mond, East Brunswick; Arunkumar R. Patel, Sommerset, all of N.J.

[73] Assignee: Bell Communications Research, Inc., Livingston, N.J.

[21] Appl. No.: 629,372

[22] Filed: Dec. 18, 1990

[51] Int. Cl.[5] H04M 3/42; H04M 3/00

[52] U.S. Cl. 379/201; 379/207; 379/269; 379/284

[58] Field of Search 379/201, 207, 96, 93, 379/94, 204, 269, 284, 396; 364/200, 300; 395/159, 160, 161, 500

[56] References Cited

U.S. PATENT DOCUMENTS

4,653,090	3/1987	Hayden	379/204
4,695,977	9/1987	Hansen et al.	379/201 X
4,782,517	11/1988	Bernadis et al.	379/201
4,785,408	11/1988	Britton et al.	364/300
4,878,240	10/1989	Lin et al.	379/67

OTHER PUBLICATIONS

"Business Services Database (BSDB): A Service Control Point (SCP) Application Designed to Support Private Virtual Network (PVN) Services," Technical Advisory TA-TSY-000460, Issue 2, Feb. 1988, Bell Communications Research, Inc.

Ljungblom, "A Service Management System for the IN", Ericsson Review No. 1, 1990, pp. 32–41.

Primary Examiner—James L. Dwyer
Assistant Examiner—Harry S. Hong
Attorney, Agent, or Firm—Leonard Charles Suchyta; Joseph Giordano

[57] ABSTRACT

A system and method for creating and modifying intelligent telephone network call processing logic trees which can be customized for individual customers and created in a user-friendly visual environment (10, 15, FIG. 3-5). Service primitives are defined as logical graph nodes (20, FIG. 2, FIG. 3) which can be visually assembled into logic trees (FIG. 5) which represent the service logic flow and which provide default values for all service options. Higher level nodes, assembled from a plurality of service primitives, can likewise be defined and stored (12, 13, 19) for later use as entities in defining yet further call processing logic trees. These call processing logic trees are interpreted to allow the service control point computers (17) to implement the services in the switched telephone network (18) by sequentially executing the specified call processing primitives. A library (12, 13, 19) of defined nodes and defined node assemblies which represent service features can thus be made available to permit graphical manipulation into complete logic trees representing new services. These logic trees are then interpreted by generic programs in the service control point to actually provide the described services.

17 Claims, 4 Drawing Sheets

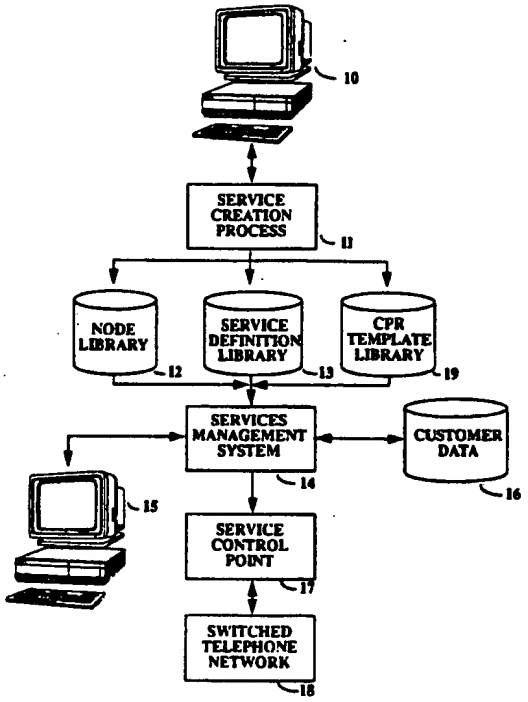


FIG. 1

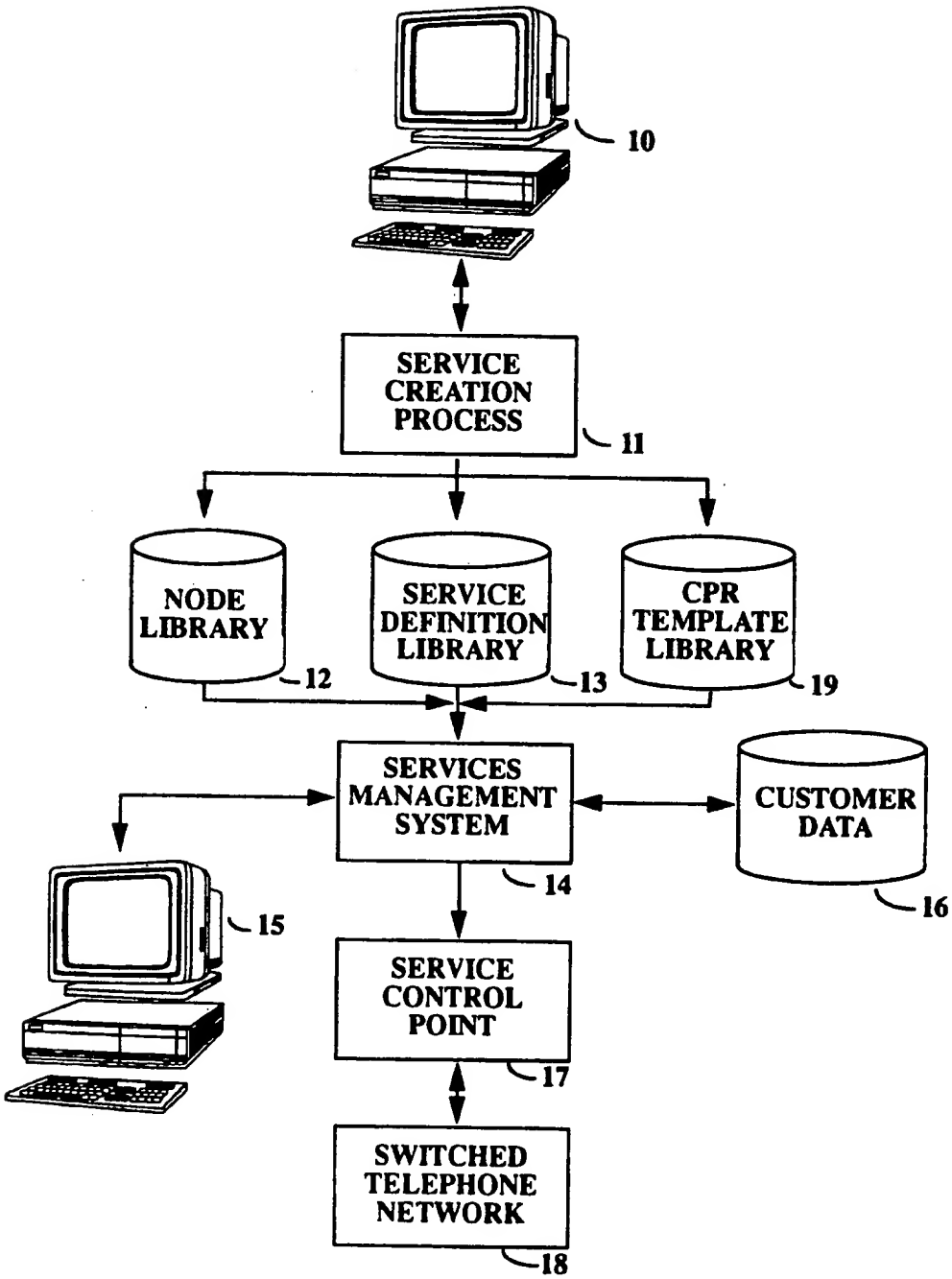

10
SERVICE CREATION PROCESS
11
NODE LIBRARY
12
SERVICE DEFINITION LIBRARY
13
CPR TEMPLATE LIBRARY
19
SERVICES MANAGEMENT SYSTEM
14
CUSTOMER DATA
16
15
SERVICE CONTROL POINT
17
SWITCHED TELEPHONE NETWORK
18

FIG. 2

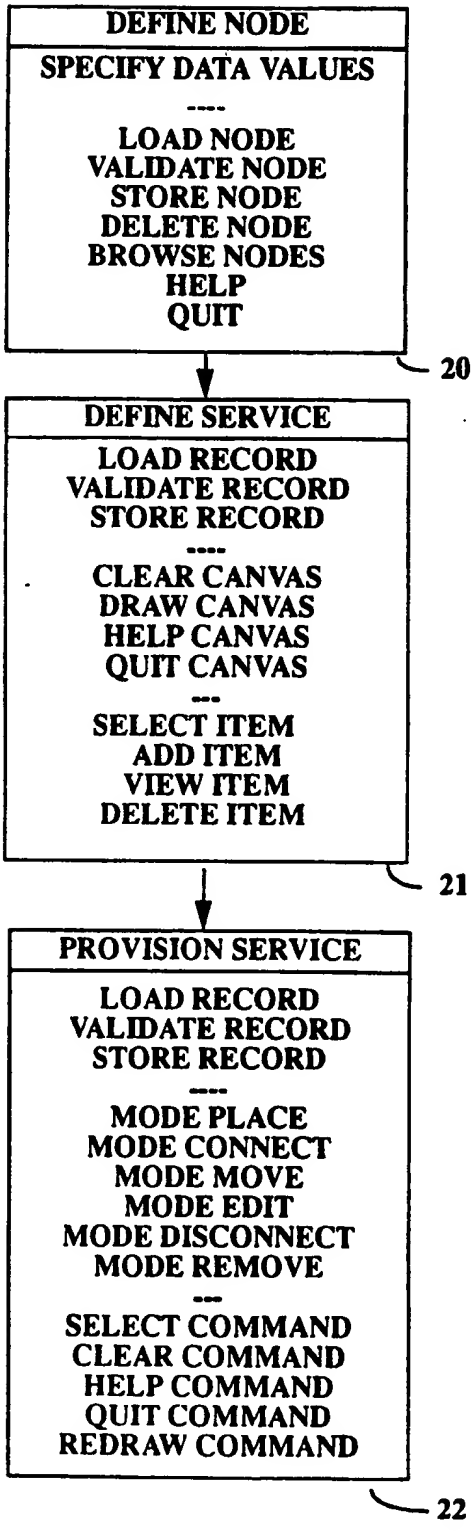


FIG. 3

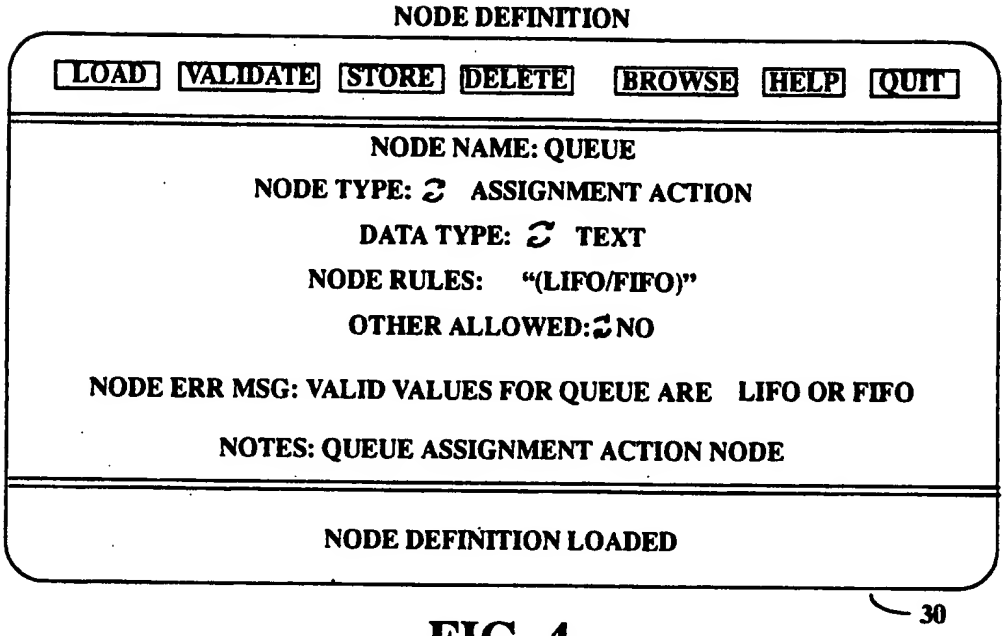

NODE DEFINITION
LOAD
VALIDATE
STORE
DELETE
BROWSE
HELP
QUIT
NODE NAME: QUEUE
NODE TYPE: ASSIGNMENT ACTION
DATA TYPE: TEXT
NODE RULES: "(LIFO/FIFO)"
OTHER ALLOWED: NO
NODE ERR MSG: VALID VALUES FOR QUEUE ARE LIFO OR FIFO
NOTES: QUEUE ASSIGNMENT ACTION NODE
NODE DEFINITION LOADED
30

FIG. 4

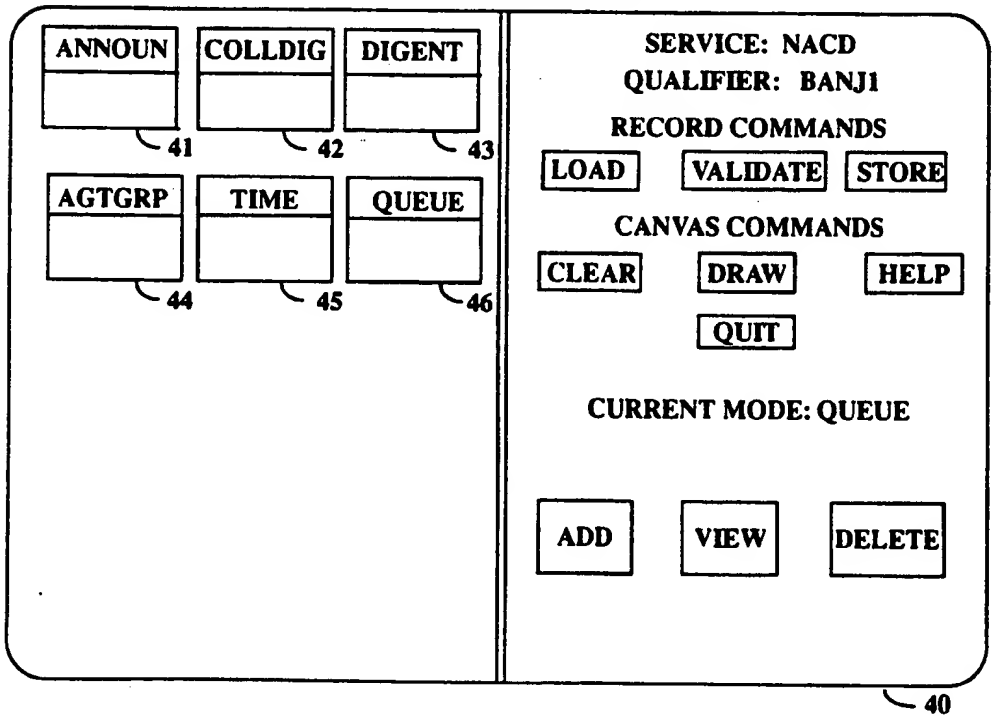
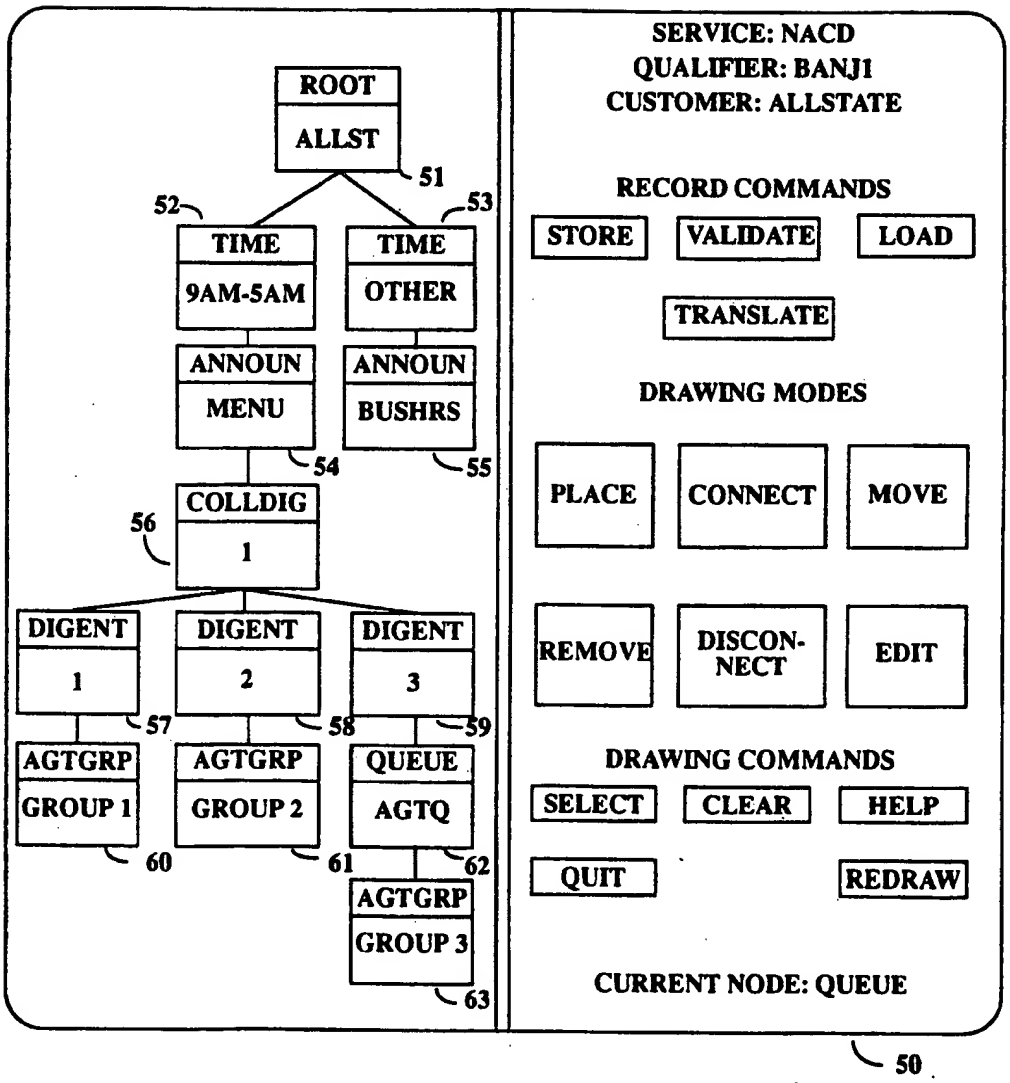

SERVICE DEFINITION
ANNOUN
COLLDIG
DIGENT
41
42
43
AGTGRP
TIME
QUEUE
44
45
46
SERVICE: NACD
QUALIFIER: BANJ1
RECORD COMMANDS
LOAD
VALIDATE
STORE
CANVAS COMMANDS
CLEAR
DRAW
HELP
QUIT
CURRENT MODE: QUEUE
ADD
VIEW
DELETE
40

FIG. 5


ROOT
ALLST
51
52
53
TIME
9AM-5AM
TIME
OTHER
ANNOUN
MENU
ANNOUN
BUSHRS
54
55
COLLDIG
1
56
DIGENT
1
DIGENT
2
DIGENT
3
57
58
59
AGTGRP
GROUP 1
AGTGRP
GROUP 2
QUEUE
AGTQ
60
61
62
AGTGRP
GROUP 3
63
SERVICE: NACD
QUALIFIER: BANJ1
CUSTOMER: ALLSTATE
RECORD COMMANDS
STORE
VALIDATE
LOAD
TRANSLATE
DRAWING MODES
PLACE
CONNECT
MOVE
REMOVE
DISCON-NECT
EDIT
DRAWING COMMANDS
SELECT
CLEAR
HELP
QUIT
REDRAW
CURRENT NODE: QUEUE
50

VISUAL PROGRAMMING OF TELEPHONE NETWORK CALL PROCESSING LOGIC

TECHNICAL FIELD

This invention relates to the creation and the provisioning of special customized telephone network services for telephone subscribers and, more particularly, to a portable, visually programmed call processing logic interface for creating and provisioning such customized services.

BACKGROUND OF THE INVENTION

New telephone services are continually being developed by telephone service providers in order to meet the needs of their customers. As such services become available, the telephone companies, subscribers and the users seek yet further improvements in these services. Special services such as 800 service, 900 service, and Private Virtual Networks (PVN) are only a few of the possible services being offered. Other services might well be conceived and might well find extensive use. Unfortunately, however, new telephone services have heretofore required long and expensive design, testing and deployment activities. Such special telecommunications services are typically provided in the public telephone network by computer program call processing sequences residing in digital switches. A typical approach to providing such special services is the introduction into the telephone network of a service implementing network element which interacts with the telephone network so as to implement the telephone services. Such service implementing network elements are variously called Service Adjuncts (SAs), Service Switching Points (SSPs) and Service Control Points (SCPs). One such Service Adjunct is disclosed in S. M. Lin and J. F. Rizzo U.S. Pat. No. 4,878,240, granted Oct. 31, 1989. A typical Service Switching Point is disclosed in J. J. Bernardis U.S. Pat. No. 4,782,517, granted Nov. 1, 1988. Finally, a typical Service Control Point is disclosed in J. O. Boese et al. application Ser. No. 453,042, filed Dec. 12, 1989. Each such service implementing network element is equipped with a set of software-implemented service primitives which can be combined in various ways to implement a number of telephone services. A set of such primitives for a Service Adjunct is disclosed in the above-mentioned Lin et al. patent. Another set of primitives for a Service Switching Point is disclosed in the above-mentioned Bernardis et al. patent. Yet another set of such primitives for a Service Control Point is disclosed in "Business Services Database (BSDB): A Service Control Point (SCP) Application Designed to Support Private Virtual Network (PVN) Service, "Technical Advisory TA-TSY-000460, Issue 2, February 1988, published by Bell Communications Research, Inc., Red Bank, N.J. All of these sets of service-implementing primitives are, as a group, generally equivalent, although each is implemented in a slightly different way.

A telecommunications network including such programmable special service implementing components is called an "intelligent network." Such networks make it possible to offer useful and profitable new services such as 800 service, Alternate Billing service and Private Virtual Network service. Intelligent Network Call Processing Logic (INCPL) is the name given to the software that "stitches together" the appropriate service primitives to enable the Intelligent Network mechanism to implement and customize such services without the need for new switch software or hardware. Currently, this INCPL is custom-developed for each new service by a service designer, usually associated with the service provider, installed in the intelligent network service implementing component and supported by a service management system for that service. Thus, instead of the software to support a new service coming from a switch vendor, it can now come from a service vendor, making it possible to reduce the interval between service concept and service offering. Unfortunately, a great deal of time is still required to design custom and implement each new service. Moreover, since such service designs are not highly portable, further delays arise due to the need to provide widely distributed software elements to support the new service in a wide variety of different environments.

SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the present invention, these and other problems are overcome by providing a user-friendly environment in which intelligent network service primitives can be assembled into telephone services utilizing the graphic capabilities of a computer workstation. If a complete telephone service is analyzed as a graph consisting of nodes and edges, the nodes represent the intelligent network service primitives executable by the service implementing components and the edges represent the order of execution of these primitives. Both action nodes and decision nodes can be defined in terms of the available service primitives and stored for later use in designing a new telephone service. Moreover, with this representation, the new telephone service can be displayed as a "tree" of such node. Such a tree display can be manipulated graphically to create and alter the logic of a service. An interpretor program, designed for a particular service implementing component, implements such display representations with the available adjunct service primitives. In accordance with this invention, telephone services can be created, manipulated and altered by simple, user-friendly graphical manipulation. Not only can intelligent network service primitives be assembled graphically into new services, but service features can be defined as assemblies of such intelligent network service primitives and, once designed, stored in a library as a single node to be invoked and reused as a single entity in designing a plurality of future services.

More particularly, graphical images or icons on a display screen are used to represent a reusable library of user-defined telephone service primitives. These images or icons can be manipulated graphically so as to assemble the images into logical trees representing new services or new service features. Once fully assembled, the electronic representation of the graphical service tree or service feature tree can be interpreted so as to actually implement the service or service feature. In this way, new telephone services can be designed and implemented much more quickly than in the prior art and thus save much of the expense previously associated with telephone service design and deployment. In addition, the extremely high level representation of telephone services implicit in the graphical service trees is an extremely portable mechanism for quickly and easily deploying such services without extensive reworking of service implementing code. Indeed, the simplicity,

speed and flexiblity of telephone service design and implementation provided with the present invention can be exploited to allow each individual telephone user to custom design telephone services exactly matching the needs of the customer.

The present invention makes possible dynamic graphical creation, customization and provisioning of new call processing services. Using both simple nodes, hereinto called primitive nodes and consisting of, primitives and complex nodes that are user-defined constructed nodes and are called user defined service feature nodes, new services can be defined rapidly and with great ease. A new service is created visually as a logic tree with nodes defined to represent a high-level abstraction of the primitive call processing actions or call processing decision points or combinations of such primitive actions and decision points. This tree, or the electronic representation of the tree, is called a Call Processing Record (CPR) since it defines exhaustively the processing necessary to provide an individual call with the defined service. Service providers, as well as subscribers themselves, can customize and provision the graphically defined service to satisfy specific customer needs by dynamically modifying and parameterizing the relevant graphical call processing record trees. The thus provisioned call processing records representing the service can then be translated from the user-friendly visual format to a rigorous program-generating format which can be generically interpreted by existing service implementing components. The distribution and installation of such call processing records to appropriate service implementing components in an intelligent network enables the thus-described service to become instantly available to telephone subscribers.

There are three steps in this service specification process: 1) defining the nodes, 2) establishing which nodes can be used in a particular service, and, 3) creating call processing records defining a particular service using the nodes that have been defined, and to which can be added the customer-specific parameters necessary to completely specify the service. These three steps can be iterated interactively to improve the quality of the resulting service, to remove "bugs," or to provide new features and capabilities.

Other aspects of a service creation and provisioning system are disclosed and claimed in the copending applications of D. L. Babson, J. A. Bajzath, Jr. and T. C. Ely, Ser. Nos. 629,371, 629,389, 629,390 and 629,447, all filed of even date herewith and assigned to applicants' assignee.

BRIEF DESCRIPTION OF THE DRAWINGS

A complete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawings, in which:

FIG. 1/shows a block diagram of an intelligent telephone network including a graphical telephone service design and deployment system in accordance with the present invention;

FIG. 2 shows a flowchart of the service definition, provisioning and implementation processes taking place in the system of FIG. 1;

FIG. 3 is a graphical representation of a display screen used to define new telephone service primitives in the system of FIG. 1;

FIG. 4 is a graphical representation of a display screen used to define the allowable subset of telephone service nodes which are used to design and implement a particular advanced telephone service in the system of FIG. 1; and

FIG. 5 is a graphical representation of a display screen used to customize a logical tree template of service primitives by specifying all of the customer-dependent variables required to implement a particular new service for a particular customer in the system of FIG. 1.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION

Referring more particularly to FIG. 1 of the drawings, there is shown a general block diagram of an intelligent network telephone service design and deployment system comprising a workstation 10 for designing new telephone services in accordance with the present invention. As will be described hereinafter, workstation 10 includes graphical facilities for defining customer special service nodes in terms of service control point primitives, bounding new special services in terms of the thus defined nodes which can be used for that service, and assembling these telephone service nodes into templates or assemblies of call processing logic units capable of providing the new telephone special services. Service creation process 11 provides the software for supporting these facilities and includes standard window processing capability as well as mouse or other visual selection apparatus support, and is typically stored in the program memory of workstation 10. Storage device 12 stores a library of the electronic representations of all of the nodes previously specified, both simple primitive nodes and more complex, user-defined service feature nodes. Storage device 13 stores the definitions of all of the services defined at workstation 10, where a service definition is simply the specification of which of the defined nodes (primitives) can be used to implement a particular service. Storage device 19 stores templates of multinode fully designed services as call processing records. In the present application, a "template" is a fully designed service, but without all of the variable parameters which are dependent on the actual customer using the service. Although shown as separate storage devices, stores 12, 13 and 19 can be contained in a single storage device.

The present invention contemplates three steps in creating new services, defining graphical nodes from which services can be assembled (node definition), prescribing which nodes can be used in a particular service (service definition) and prescribing the logical interrelationships of the various nodes necessary to produce the desired service (call processing record creation). As new nodes are defined, they are stored in node library 12 for later retrieval and reuse. Once a new service is fully defined in terms of primitive or complex service nodes, a representation of that service definition is stored in a service definition storage facility 13. Once the service nodes are assembled into a service logic tree or template, that template is stored as a call processing record (CPR) in call processing record template library 19. The templates in library 19 require only the specification of certain customer-dependent variables required to fully implement the service for a particular customer.

A services management system 14 provides administrative control over a plurality of service control points such as service control point 17 which actually imple-

ment the new services by exchanging network control messages with the switched telephone network 18. The details of the service control point 17, along with its interaction with the switched telephone network 18, are detailed in the aforementioned copending application of J. O. Boese et al. Since the structure and operation of the service control point 17 and the switched telephone system 18 comprise well-known telephone facilities, they will not be further described here. Sufficient to note that these elements are able to implement telephone network service primitives which can be utilized to realize telephone services and which are similar or identical to those disclosed in the aforementioned J. J. Bernardis and S. M. Lin patents and BSDB publication.

Associated with the services management system 14 is a customer data base 16 which contains detailed information concerning the customers utilizing telephone network 18. Also associated with services management system 14 is a further computer workstation 15 which can be used to add customer preferences and other customer specific data to the call processing record templates defined in store 19. This addition of the customer specific data to a call processing record template is called "service provisioning" and permits the service templates passed to service control point 17 to be customized for a particular customer or set of customers using network 18. Workstation 15 can, of course, be combined with workstation 10 if the two workstations are at the same location.

The workstations 10 and 15 can be any modern workstation supporting graphical manipulation capability. One such workstation is the SUN 3/160 terminal running under the SUN operating system and using the SUNVIEW® graphics environment. This environment provides window support, mouse control, and graphic creation and manipulation capability adequate to implement the present invention. Many other modern workstations, however, would likewise support implementations of the present invention.

The present invention will be better understood by considering the flowchart of FIG. 2. In FIG. 2 there is shown a flowchart of the process for designing and deploying new services for telephone subscribers in accordance with the present invention. Three major steps are involved, node definition, service definition and call processing record creation. These steps will be taken up individually below.

The intended user of the process of FIG. 2 is the service creator, typically the telephone service provider. As suggested in box 20 of FIG. 2, this person must create the nodes to be used in new services. The new nodes are created dynamically by specifying the following properties of the new node.

1. Node Name
2. Node Type
3. Data Type
4. Node Rules
5. "Other" Allowed
6. Node Error Message
7. Notes

As is shown in detail in FIG. 3, a node definition screen can be used to capture the node properties as data items. The "Load," "Validate," "Store," "Delete," "Browse," "Help," and "Quit" command buttons are utilized to access and process nodes as entities. The data acquisition fields are used to define new nodes by specifying the node name and the node properties. Using these properties, a node can be administered by the service management system 14 of FIG. 1 and implemented by the service control point 17 of FIG. 1 without the necessity of writing new service implementing code. As illustrated in FIG. 3, the user is presented with an on-screen form which requires that the user specify the values for each of these properties. The cursor initially is located at the first data field (NODE NAME:) and advances to successive data fields as the <enter> or <return> key is depressed. Once these properties have been specified, the user can store the completed node definition for the newly created node in a node library in store 12 (FIG. 1), using the command keys identified in the top portion of the screen display of FIG. 3.

The properties captured by the node definition screen of FIG. 3 contain key information about the new node and, when necessary, can be translated and sent to the service control point 17 of FIG. 1 to drive a generic Call Processing Record (CPR) interpreter (such as that disclosed in the aforementioned Bernardis patent), so that it can support the use of this new node in any call processing records that are created and provisioned for services. Additional or supplemental properties, such as associated service control point call variables, node connection rules and node translation rules, can be added to the node library at a later time to completely characterize the meaning and intent of these operations at a particular services management system 14 and a particular service control point 17 (FIG. 1) in a generic manner. More particularly, a specific identification of the service implementing primitive in one or more service implementing network components which can be used to realize the node can be explicitly added to the node definition. The presence of such an identification of the actual code sequence in the service control point would substantially simplify the design of the interpretor program which translates the call processing record into actual service implementation. In this sense, the nodes become elements of an extendible, high level programming language for specifying telephone services.

In response to the NODE NAME prompt, the user supplies any unique name chosen for the new node. This name will hereafter be used to identify the node. This new node name can be thought of as a higher-level abstraction representing a specific operation using predefined call processing variables. Nodes can be either decision nodes or action nodes. The following node types, supplied in response to the NODE TYPE prompt, are possible:

TABLE 1

NODE TYPES	
ACTION NODES	DECISION NODES
Assignment Action	Branch Decision
Boolean Operation Action	Integer Decision
Collect Action	Percent Decision
Connect Action	String Decision
Concatenate Action	
CPR Access Action	
Diagnostic Action	
Return Action	
Table Access Action	
Temporary Hand Over Action	
Terminate Action	

The node type selected provides information that can be used during call processing record validation and call processing record code generation. These node

types correspond to all of the basic types of actions and decisions that might be utilized in providing telephone special services. They are already programmed or programmable into prior art service implementing network components such as service control point 17. Should a new type of node be required, however, it is necessary to specify the new type, and to provide generic implementation of the activity of the new node type in the service control point 17.

The node types have particular meaning to service control point 17, which must execute the call processing record in real time. Specifically, the node type informs service control point 17 what functions or call primitives are to be invoked. These node types thus represent the "programmed capabilities" of service control point 17, which, when combined with the node name, representing a particular pre-defined call variable in the service control point call processing execution environment, enable service control point 17 to actually perform the requested operations using the correct data.

The DATA TYPE field specifies the format of the data that will be acceptable as input values for this node when this node is used in a call processing record. The data formats allowed are decimal numbers, alphanumeric text, telephone numbers, trunk identifiers, variable data, and billing information. The data type is used during call processing record validation at the time the call processing record is "provisioned" by the specification of customer-dependent data values.

To administer a call processing record, services management system 14 of FIG. 1 must be able to determine whether valid values have been entered for the nodes in the call processing record tree. Therefore the user must specify the rules to be used when validating the input values for this node when this node is used in a call processing record. In accordance with the present invention, a regular expression can be used to represent the validation rules. Moreover, such validation rules can be entered dynamically in the node definition as opposed to hard coding them in a program. For example, to specify the validations for a queuing node, the user can specify the regular expression " (lifo|fifo)$". This expression states that the node can accept either "last in, first out" or "first in, first out" queue behavior as input values.

In response to the OTHER ALLOWED, prompt, the user must specify whether the node can accept the word "other" as an input value. For example, if a day of the week node were defined and available for a particular service, a call processing record would be constructed with the day of the week node in the tree. If the input values for the day of the week node were specified as "Monday and Tuesday," then a second branch would need to be created to cover what should be done on Saturday, Sunday, Wednesday, Thursday and Friday. Rather than spell out the remaining days, the string "other" could be used. For some nodes "other" is not a sensible value. Therefore, during node definition, the user must indicate whether "other" is a valid input value or whether entering "other" should result in a validation error.

In response to the NODE ERROR MESSAGE prompt, the user enters a string to be printed when an inappropriate input value is specified for a node during the provisioning of a call processing record. This allows "on the fly" validation of data entries during all succeeding uses of this node definition.

The NOTES prompt permits a convenience field for the user. It can be used to note limitations on the node behavior, a general functional description of the node or any other information the user wishes to associate with the node definition.

Also shown at the top of screen 30 of FIG. 3 are a series of command light buttons labeled "Load," "Validate," "Store," "Delete," "Browse," "Help," and "Quit." These command buttons are operated, for example, by a mouse-driven cursor, to accomplish the associated screen display control functions. For example, "Load" will load any particular previously existing node definition (e.g., for editing) simply by giving the node name and evoking the "Load" command. Similarly, "Validate" performs the validation on the data entries made in response to the prompts. "Store" stores the newly defined node in the node library 12 of FIG. 1. The "Delete" command deletes the identified node definition from library 12. "Browse" allows the user to browse through all of the previously defined nodes, "Help" provides screens of useful ancillary information likely to be of assistance to the user during the node definition activity, and "Quit" allows the user to terminate the node definition activity. The implementation of these so-called "command buttons" are well-known in the art and will not be further described here.

Returning to FIG. 2, the second box 21 represents the activity of defining a new service in terms of previously defined nodes. In this context, a "record" is a service definition and box 21 includes "Load," "Validate" and "Store" commands for these service definition records. In addition, box 21 includes "Canvas" commands. These commands permit the visual and graphical manipulation of previously defined nodes into service definitions. As can be better seen in FIG. 4, the left half of the screen 40 is the drawing area where visual representations of previously defined nodes can be assembled into service definitions. The "Canvas" commands include "Clear" (to clear the canvas area), "Draw" (to prepare for manipulating items on the canvas), "Help" (to obtain screens of useful information to assist in the use of the system), and "Quit" (to terminate the service definition session). In addition, individual items in the canvas area can be manipulated with the commands "Select" (to select a particular node), "Add" (to add a selected node to the service definition), "View" (to view the node definition of the selected node), and "Delete" (to delete a node from the service definition).

The intended user of the service definition process is the service creator. This creator utilizes the service definition process to define the subset of nodes in the node library that are to be used in a specific service. The user is presented with the form shown in FIG. 4 which enables the user to specify the nodes from the node library that are allowed for use by a specific service with a certain qualifier. A qualifier could, for example, be a specific area of service in the service provider's territory. With the functions described above, the use of any particular node in a service can be defined by a service creator and administered by the service management system 14. Once the allowable node names for a particular service have been specified, using the screen 40 of FIG. 4, the user can store the service definition for the new service in service definition library 13 (FIG. 1). These service definitions can thereafter be edited, augmented or deleted from the service definition library 13 whenever desired.

The service creation user can distribute the node library and the service definition library in store 13 to the generic service management system 14 for use in administering the new service. Services management system 14 will permit the use of only the specified subset of nodes in all call processing records that are created and provisioned for this particular service. Additional properties, such as service-specific node connection rules and node translation rules, can be added to the service definition tables at this time to completely and generically characterize the meaning and intent of these operations for the service management system 14.

Returning to FIG. 2, the box 22 represents the procedures for "provisioning" the service defined in box 21. In this regard, the term "provisioning" means creating the logic tree representing the new service and customizing the defined service in terms of node parameter values. The screen 50 of FIG. 5 illustrates a mechanism for carrying out this process. The service creator can use this process to define a default representation of the service offering as a template of nodes interconnected in a call processing record tree format. The allowed nodes for a particular service are the ones specified through the service definition process. The service provisioning user can use this process to customize the default service template and to provision the service with node parameter values. The user is presented with the visual programming tool illustrated in FIG. 5 to enable a graphical, user-friendly specification and input of the appropriate service customization and provisioning information, based on service, qualifier and customer key information. The qualifier could be, for example, a specific area of service in the service provider's territory. The screen of FIG. 5 provides record manipulation functions (Load Record, Validate Record, Store Record, and Translate Record), node manipulation functions (Place, Connect, Move, Edit, Disconnect, and Remove), and draw commands (Select, Clear, Help, Quit, and Redraw).

With the above described functions, the user can define a service template of nodes interconnected in a call processing record tree format as a default representation of the service offering. Furthermore, the true power of the present invention becomes evident when it is realized that the service creator can define modular templates of common reusable node-based logic as service features, which can then be added to the node library as nodes for reuse by other features and services. In effect, the present invention provides a mechanism for using the set of "programmable capabilities" available at the service control point 17 and define reusable, user-friendly and high-level modules that enable easier and faster visual programming of the call processing logic.

It should be noted that assemblies of low level nodes can be associated together into a higher level capability which, once given a name, can be retrieved, manipulated and used in creating new services just like a low level node. In this way, as time proceeds, the creation of new services becomes easier and easier since ever higher levels of service primitives become available for assembly into the new services.

Once the default service template has been specified, the user can store the service template definition for the new service in service template definition store 19. The user can thereafter distribute these service template definition tables to the generic services management system 14 for use in administering the new service, providing a default service logic flow for the purpose of provisioning. Furthermore, the availability of service node definitions in store 13 will ensure that provisioning will support the use of only the creator-specified subset of nodes in any call processing records that are customized and provisioned for this particular service.

The service provider or the subscriber can also use the process illustrated in FIG. 5 to customize the default service template and to provision the service by providing node parameter values. At this stage, the node values supplied are validated dynamically by invoking the node validation rules specified for each node name in the node library. Additional node and service definition properties, such as node connection rules and node translation rules, can be specified to drive the validation and translation operations at the services management system 14 in a generic manner.

The attached Appendix provides psuedocode for implementing the flowchart of FIG. 2. With this pseudocode and the description herein provided, any person of ordinary skill in the programming art is able to fully implement the present invention.

It should also be clear to those skilled in the art that further embodiments of the present invention may be made by those skilled in the art without departing from the teachings of the present invention.

APPENDIX
Pseudocode

Node Definition Process Pseudocode

```
invoke ('node_definition')
do until (invoke ('quit'))
  if (view (node_definition))
    then input (node_name)
      invoke ('load')
      if (node_definition (node_name) in library
        then
          retrieve (node_definition (node_name))
          display (node_definition (node_name))
        else
          create (node_definition_window (node_name))
          display (node_definition_window (node_name))
      endif
      end_invoke
  endif
  if (specify (node_properties))
    then
      set_default (field_values (node_name))
      enter (field_values (node_name))
    else
      if (modify (node_properties))
        then update (field_values (node_name))
      endif
  endif
  if (validation_check (node_name))
    then
      invoke ('validate')
        verify (node_name, field_entries, field_entry_
        combinations)
      end_invoke
  endif
  if (store (node_definition))
    then
      invoke ('store')
        retrieve (node_definition (node_name))
        overlay (node_definition, node_properties)
        store (overlay (node_name))
      end_invoke
  endif
  if (delete (node_definition))
    then
      invoke ('delete')
        retrieve (node_definition (node_name))
        delete (node_definition (node name))
      end_invoke
  endif
```

-continued

APPENDIX
Pseudocode

```
  if (browse (node_definition)
    then
      invoke ('browse')
        display (node_properties)
        retrieve (node_definitions)
        display (node_definitions)
      end_invoke
  endif
end_do
```

Service Definition Process Pseudocode

```
invoke ('service_definition')
do until (invoke ('quit'))
  if view (service_definition)
    then
      input (service_name, service_qualifier)
      invoke ('load')
        retrieve (service_definition (service_name, service_
        qualifier))
        display (node_name_list)
      end_invoke
  endif
  if view (all_nodes)
    then
      invoke ('select')
        generate (menu (all_nodes))
      end_invoke
  endif
  if (select_node)
    then
      invoke ('select')
        select (desired_node, menu (all_nodes))
        display (desired_node, current_node_display)
      end_invoke
  endif
  if use (current_node)
    then
      invoke ('add')
        position (current_node, canvas)
        add (current_node, service_definition)
      end_invoke
  endif
  if disallow (current_node)
    then
      invoke ('delect')
        select (current_node)
        delete (selected_node)
      end_invoke
  endif
  if view (current_node_definition)
    then
      invoke ('view')
        select (current_node)
        display (node_definition, current_node)
      end_invoke
  endif
  if validate (service_definition)
    then
      invoke ('validate')
        verify (field_entries, service_definition)
        verify (field_entry_combinations, service_definitions)
      end_invoke
  endif
  if store (service_definition)
    then
      invoke ('store')
        retrieve (service_definition (service_name, service_
        qualifier))
        overlay (service_definition (service_name, service_
            qualifer), canvas_nodes))
        store (service_definition (service_name, service_qualifier))
      end_invoke
  endif
  if clear (canvas_area)
    then
      invoke ('clear')
        clear (all_nodes, canvas_area)
      end_invoke
  endif
  if redraw (canvas_area)
    then
      invoke ('draw')
        clear (all_nodes, canvas_area)
        auto_invoke ('load')
      end_invoke
  endif
end_do
```

CPR Input Process Pseudocode

```
invoke ('CPR_input')
do until invoke ('quit')
  if view (existing_CPR_tree)
    then
      input (service_name, qualifier_name, customer_name)
      invoke ('load')
        retrieve (node_names (service_name, service_qualifier))
        retrieve (node_definitions (node_names)
        retrieve (CPR_tree (service_name, service_
            qualifier, customer_name))
        display CPR_tree (service_name, serice_qualifer,
            customer_name))
      end_invoke
  endif
  if view (allowed_nodes (service_name, service_qualifier))
    then
      invoke ('select')
        generate (menu (allowed_node_names))
      end_invoke
  endif
  if select (allowed_node)
    then
      invoke ('select')
        identify (desired_node)
        display (desired_node, current_node)
      end_invoke
  endif
  if add (current_code, CPR_canvas)
    then
      invoke ('place')
        position (current_node, CPR_canvas)
        place (current_node, CPR_canvas)
      end_invoke
  endif
  if connect (node_name1, node_name2, CPR_canvas)
    then
      invoke (connect)
        select (from_node)
        select (to_node)
        draw_line (from_node, to_node
      end_invoke
  endif
  if move (node_name)
    then
      invoke (move)
        erase (node_name, old_location
        redraw (connections, old_location, new_location)
      end_invoke
  endif
  if edit (node_name_node_value)
    then
      invoke ('edit')
        select (node_name)
        display (edit_menu)
        input (node_value)
        invoke ('enter')
          erase (old_value)
          display (new_value)
        end_invoke
      end_invoke
  endif
  if disconnect (node_name)
    then
      invoke ('disconnect')
        select (node_name)
        erase_connection (node_name, node_parent)
      end_invoke
  endif
  if remove (node_name)
    then
      invoke ('remove')
```

-continued

APPENDIX
Pseudocode

```
        select (node_name)
        remove (node_name, children, connections)
      end_invoke
  endif
  if validate (CPR_node)
    then
      invoke ('validate')
        for (each node in CPR_tree)
        do
          validate (node_value, node_rule)
        end_do
        display (error_messages)
        correct (node_values, connections)
      end_invoke
  endif
  if store (CPR_tree)
    then
      invoke ('store')
        retrieve (CPR_tree (service_name, service_qualifier,
            customer_name))
        overlay (CPR_tree (service_name, service_qualifier,
            customer_name), CPR_tree (canvas_area))
        store (CPR_tree, overlay)
      end_invoke
  endif
  if clear (canvas_area)
    then
      invoke ('clear')
        erase (canvas_area)
      end_invoke
  endif
  if redraw (canvas_area)
    then
      invoke ('redraw')
        erase (canvas_area)
        auto_invoke ('load')
      end_invoke
  endif
end_do
```

What is claimed is:

1. A method for visually programming telephone services for implementation in a telephone network having a service control point, said method comprising the steps of

defining new nodes of network service primitives for storage in a storage means,

recalling from said storage means previously defined nodes,

assembling a set of said defined new nodes and recalled nodes to be used in one of said telephone services wherein said assembled set of nodes can be stored as a service definition in a service definition library,

graphically interrelating said assembled nodes into a logical graph representing a telephone service call processing sequence,

storing said logical graph in a logical graph template library within said storage means to be used in providing the defined telephone service,

retrieving a logical graph template from said template library when said telephone service is to be provisioned for a customer, and

specifying customer dependent variables in said logical graph template for provisioning customer service

wherein said logical graph template with said specified customer dependent variables can be sent to a service control point for interpretation by an interpretive process to provide service.

2. The method according to claim 1 wherein said step of defining new nodes comprises the step of specifying the name of one of said new nodes.

3. The method according to claim 2 wherein said step of defining new nodes comprises the step of specifying the node type of said one new node.

4. The method according to claim 2 wherein said step of defining new nodes comprises the step of specifying the data type of parameter values associated with said one new node.

5. The method according to claim 4 wherein said step of defining new nodes further comprises the steps of specifying validation rules for parameter values associated with said one new node, and specifying an error message to be displayed in response to the failure of either said data type or said parameter values to satisfy said validation rules.

6. The method according to claim 1 wherein said step of defining new nodes comprises the step of displaying selected ones of the defined new nodes.

7. The method according to claim 1 wherein said step of graphically interrelating includes the step of adding either a new or previously defined node to said call processing sequence.

8. The method according to claim 1 wherein said step of graphically interrelating includes the step of deleting either a new or previously defined node from said call processing sequence.

9. The method according to claim 1 wherein said step of graphically interrelating includes the step of interconnecting two of said set of assembled nodes in said call processing sequence.

10. The method according to claim 1 wherein said storage means comprises:

a node library for storing said new and previously defined nodes, said service definition library and said template library.

11. The method according to claim 1 wherein said logical graph can be defined as a complex node and stored for later use and recall in said node library.

12. A system for defining software implemented services in a telephone network having a service control point and programmable facilities of having executable service primitives, said system comprising:

a node data store for storing groupings of said service primitives as primitive nodes;

a service definition data store;

a call processing record template store;

a graphical user input and display device; and

service creation process means for retrieving from said node data store any number of said nodes and displaying said nodes on said graphical user input and display device as a graphical symbol wherein a user using said graphical user display device can select from said nodes a set for defining a service and store said set of nodes together as a service definition in said service definition data store, manipulate said set of nodes into a graphical abstraction of a logical service process, and store said graphical abstraction of a logical service process as a call processing record in said call processing record template store, and wherein customer data can be added to said call processing record template store and sent to said service control point to provision services for a customer.

13. The system according to claim 12 wherein said service creation process means further comprises means for defining at least one new node of service primitives

by specifying a node name, a node type, and a new set of node properties comprised of parameter values and means for storing said new node in said node data store.

14. The system according to claim 13 wherein said service creation process means further comprises means for modifying said nodes by changing the values for properties that make up said nodes.

15. A system for defining software implemented services in a telephone network having programmable facilities consisting of executable service primitives, said system comprising:

a node data store for storing groupings of said service primitives as primitive nodes;

a service definition data store;

a call processing record store;

a graphical user input and display device; and

service creation process means for retrieving from said node data store any number of said nodes and displaying said nodes on said graphical user input and display device as a graphical symbol, said service creation process means comprising:

means for defining at least one new node of service primitives by specifying a node name, a node type, and a new set of node properties comprised of parameter values said means for defining at least one new node further comprising;

means for specifying an error message to be displayed in response to the failure of either said node type or said parameter values to satisfy validation rules; and

means for storing said new node in said node data store,

wherein a user using said graphical user display device can select from said nodes a set for defining a service and store said selected set of nodes defined as a service in said service definition data store, manipulate said set of nodes into a graphical abstraction of a logical service process creating a call processing record logic tree and store said call processing record logic tree in said call processing record data store, and wherein said user can add to said graphical abstraction of a logical service process customer data creating a customized call processing record to be sent to a service control point to be executed when service is requested.

16. The system according to claim 15 wherein said service creation process means further includes means for adding a node from said node data store to said call processing record logic tree.

17. The system according to claim 16 wherein said service creation process means further includes means for deleting a node from said call processing record logic tree.

* * * * *